# MSX

# BASIC MSX et MSX-DOS

Bruno VANRYB
Roger POLITIS



EYROLLES

# MSX

## BASIC MSX et MSX-DOS

## CHEZ LE MEME EDITEUR

| | |
|---|---|
| AUBERT et SCHOMBERG | - *Pratiquez l'intelligence artificielle* - 144 p. ; 1985, (coll. Micro-ordinateurs). |
| DELANNOY | - *Initiation à la programmation* - 192 p. ; 1984. |
| DELANNOY | - *Apprendre à programmer en Basic* - 272 p. ; 1984. |
| DELANNOY | - *Les fichiers en Basic sur micro-ordinateur* - 168 p. ; 1984, (coll. Micro-ordinateurs). |
| DE GEETER | - *FORTH pour micros* - 192 p. ; 1984, (coll. Micro-ordinateurs). |
| HIRSCH | - *Le Basic facile par une méthode progressive* - 288 p.; 1984, (coll. Pratique de l'Informatique). |
| HENRIC-COLL | - *Le FORTH en douceur* - 160 p. ; 1984. |
| KRUTCH | - *Expériences d'intelligence artificielle en Basic* - 128 p. ; 1984, (coll. Micro-ordinateurs). |
| MONTEIL | - *Premiers pas en LOGO* - 104 p. ; 1984, (coll. Micro-ordinateurs). |
| DE ROSSI | - *Apprentissage rapide du Basic* - 216 p. ; 1984, (coll. Pratique de l'Informatique). |
| VULDY | - *Graphisme 3D sur votre micro-ordinateur* - 128 p. ; 1984, (coll. Micro-ordinateurs). |

# MSX

## BASIC MSX et MSX-DOS

par

**Bruno VANRYB** et **Roger POLITIS**

EYROLLES

61, boulevard Saint-Germain – 75005 Paris
1985

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

# Introduction

## UN STANDARD DE HAUT NIVEAU

*Jusqu'à présent, le seul point commun permettant de caractériser les ordinateurs dits "familiaux", c'est l'impossibilité de transférer des programmes écrits spécialement pour un micro-ordinateur sur d'autres machines. Cette incompatibilité chronique présente un inconvénient majeur pour l'utilisateur: si les prévisions de vente du constructeur s'avèrent erronnées, les programmeurs professionnels ne s'intéressent pas à la machine, faute de débouché sérieux. Il ne reste plus à l'acquéreur malchanceux qu'à apprendre la programmation s'il veut profiter des possibilités de son ordinateur. Bien entendu, s'il change de matériel, il devra non seulement racheter tous les programmes qu'il possède, mais*

*également reécrire ceux qu'il a réalisés lui-même! Dans le domaine professionnel, l'IBM PC à mis fin à une situation du même type, tous les constructeurs de machines 16 bits s'alignant plus ou moins sur ce standard. L'apparition du standard MSX (Microsoft Super eXtended basic), dont les spécifications ont été définies par MicroSoft pour un groupe de 16 constructeurs japonais constitue la première tentative de standardisation appliquée à l'informatique familiale.*

*L'intérêt d'un standard, c'est qu'il soit de qualité. Dans le cas du MSX, cette qualité peut se résumer en quelques lignes:*

— *Le cahier des charges est suffisamment précis pour garantir une compatibilité* totale *entre toutes les machines MSX.*

— *Le MSX-BASIC, est tout simplement le Basic graphique de l'IBM PC, que MicroSoft a adapté à une utilisation familiale. C'est la première fois qu'un langage de niveau professionnel est implanté sur un micro grand public!*

— *Le clavier est de type machine à écrire, et ne fera pas regretter les touches en gomme molles très à la mode actuellement.*

— *Toutes les machines sont livrées avec un ou deux lecteurs de cartouches permettant l'utilisation des logiciels sans passer par le support cassette, lent et peu fiable.*

— *Le standard prévoit neuf interfaces, parmi lesquels une sortie pour imprimante parallèle. Tous les périphériques de tous les constructeurs pourront être transférés d'une machine à l'autre.*

— *La bibliothèque de logiciels disponibles est considérable, puisque 500 programmes sous forme de cartouches enfichables sont déjà disponibles en langue anglaise.*

— *Un Basic étendu lié à un système d'exploitation dérivé du MS-DOS permettra l'emploi de lecteurs de disquettes (MSX-DOS).*

*Si l'on ajoute à ces caractéristiques étonnantes la mise en vente des machines à des prix tout à fait comparables à ceux des micro-ordinateurs concurrents, on mesure le caractère révolutionnaire de ce standard, qui pourrait bien ouvrir une nouvelle ère de l'informatique familiale...*

## UN MANUEL DE RÉFÉRENCE

*Basé sur la référence officielle du MSX telle qu'elle a été définie par MicroSoft, ce livre regroupe toutes les instructions du MSX-BASIC, illustrées par de nombreux programmes commentés, ainsi que les*

*commandes du MSX-DOS, le système d'exploitation de disquettes du standard. Le classement des instructions par familles permettra à l'utilisateur chevronné de retrouver rapidement les informations dont il a besoin. Le débutant n'a pas été oublié, la première partie de l'ouvrage incluant un cours de programmation portant sur les instructions les plus simples du langage. L'intérêt d'un ouvrage spécialisé MSX réside dans sa portabilité : Toutes les informations et programmes contenus dans ce livre s'appliquent à toutes les machines correspondant au standard.*

# Table des matières

# Table des matières

## 1.1. GÉNÉRALITÉS

Un microprocesseur est programmable avec des instructions élémentaires très simples qui permettent de manipuler des informations binaires, formées d'une succession de 0 et de 1. Afin de rendre la programmation accessible à un utilisateur non technicien, on utilise des langages de programmation dits "évolués". Très proches du langage parlé, ceux-ci utilisent un vocabulaire codifié basé sur l'anglais. Chaque instruction est traduite en code machine par un logiciel interne spécialisé. Pour le MSX-BASIC, ce programme est appelé **interpréteur Basic**. Intégré dans une mémoire spécialisée appelée **mémoire morte** (ROM en anglais), il a été écrit une fois pour toutes et ne peut être modifié. Il est identique sur toutes les machines **MSX**, ce qui facilite la portabilité des programmes d'un ordinateur à l'autre. La syntaxe du Basic se compose d'un ensemble d'**Instructions**, de **Fonctions**, d'**Opérateurs** et d'**Opérandes** ou **Arguments**. Dans ce chapitre, une première explication de ces termes est offerte. Nous conseillons au lecteur de ne pas négliger ces connaissances de base auxquelles nous ferons fréquemment référence dans la suite de ce livre.

### 1.1.1. Instructions et fonctions

— Une **Instruction** est un mot-clé compris par le Basic, qui implique l'exécution d'une tâche précise par l'ordinateur. Certaines instructions doivent être suivies d'un argument, d'autres non.

***Exemples*** :

Dans PRINT "BONJOUR", PRINT est l'instruction et "BONJOUR" est l'argument. Mais PRINT peut aussi être utilisé seul.

Par contre, CLS (Effacement de l'écran) est une instruction utilisée toujours seule, alors que INPUT implique nécessairement l'emploi d'un ou plusieurs arguments.

— Une **Fonction** est également un mot-clé, mais il s'agit plutôt d'une "question" que l'on pose à l'ordinateur : celui-ci doit nous retourner une valeur en réponse. Une fonction s'utilise toujours à la suite d'une instruction, en général PRINT ou LET.

***Exemples*** :

Dans PRINT SIN(PI/2), SIN est la fonction et PI/2 son argument.

LET A=HEX$(100) donnera à la variable A la valeur hexadécimale de l'argument, ici le nombre 100.

### 1.1.2. Variables et constantes

Les **Opérandes** sont les valeurs, nombres ou séquences de caractères servant d'argument aux instructions et fonctions. On note deux types d'opérandes, les opérandes numériques et les opérandes littérales :

— Dans la première catégorie, on peut citer les **constantes**, qui sont des nombres pouvant être soit entiers (100, 20), soit décimaux (3.14, 255.33) ou plus généralement des réels ($2.2\ 10^3$).

  Les **variables** peuvent, contrairement aux constantes, changer de valeur au cours d'un programme. Celle-ci peut être définie par l'utilisateur ou représenter le résultat d'un calcul. L'emploi de variables suit des règles particulières dévelppées au chapitre 2.1.

— Les opérandes littérales ne sont plus considérées comme des nombres, mais comme des suites de caractères. Il s'agit des **chaînes de caractères**, elles aussi subdivisées en constantes et variables :

  Une constante de chaîne est une suite quelconque de caractères pouvant aller jusqu'à 255, toujours incluse entre guillemets.

  "PROGRAMME", "TOTO" sont des constantes de chaîne.

  Comme pour les variables numériques, les variables chaînes doivent être définies par l'utilisateur et peuvent changer de valeur en cours de programme. Leur mode d'emploi est détaillé au paragraphe 2.1.

### 1.1.3. Les opérateurs

Les **Opérateurs** sont de trois types :

1) Les opérateurs arithmétiques, que tout le monde connaît bien : +, —, * (multiplié par, / (divisé par), et ^ (élévation à une puissance).

2) Les opérateurs relationnels, >, >=, <, <=, =, <> (supérieur, supérieur ou égal, inférieur, inférieur ou égal, égal, différent) qui servent à comparer des nombres et des chaînes.

3) Les opérateurs logiques, AND, OR, NOT, XOR, EQV, IMP qui s'utilisent pour combiner des expressions utilisant des opérateurs relationnels.

## 1.2. PREMIER CONTACT

### 1.2.1. La mise en route

Une fois l'ordinateur sous tension, l'écran est initialisé avec le message suivant :

```
MSX BASIC Version 1.*
Copyright 1983 by Microsoft
12431 Bytes free
Ok
```

Ce message indique que l'ordinateur est prêt à recevoir une commande à partir du clavier. Le curseur clignotant visualise l'endroit où le texte s'affichera lors de cette entrée.

Avant de commencer à parler de programmation, il faut repérer la touche RETURN qui sert à valider toute instruction ou commande après son entrée. Sur tous les MSX, elle est repérée par le même symbole :



### 1.2.2. Le mode direct et mode programme

Toute instruction ou séquence d'instructions entrée au clavier est exécutée immédiatement dès que la touche RETURN est actionnée : C'est le mode direct. On peut dans ce mode tester une instruction ou une fonction, ou encore effectuer un calcul dont le résultat s'affiche directement à l'écran.

***Exemples*** :

```
PRINT "BONJOUR"  [RETURN]  écrit le mot "BONJOUR" à l'écran.
PRINT 5*2        [RETURN]  écrit le résultat (10) à l'écran.
```

Le mode direct ne permet pas de conserver un résultat ou d'exécuter automatiquement une suite d'instructions. Pour ce faire, il est nécessaire d'écrire un **Programme**. Toute suite d'instructions précédée d'un numéro est considérée par le Basic comme une ligne de Programme. Le fait de frapper RETURN n'exécutera pas les instructions, mais les **mémorisera** pour une utilisation ultérieure : C'est le mode "différé" ou mode programme.

On appelle **programme Basic** une suite de lignes numérotées contenant une ou plusieurs instructions du langage. Une instruction spécia-

lisée, **RUN** sert à exécuter le programme en suivant l'ordre de la numérotation.

***Exemple*** : Entrer successivement :

```
10 CLS                              [RETURN]
20 PRINT "JE SUIS"                  [RETURN]
30 PRINT "UN PROGRAMME"             [RETURN]
40 PRINT "ECRIT EN BASIC"           [RETURN]

RUN                                 [RETURN]
```

Ce petit programme efface l'écran (ligne 10), puis imprime successivement les trois messages contenus dans les trois lignes 20, 30 et 40.

### 1.2.3. La ligne de programme

Certaines règles de base doivent être respectées lors de l'entrée des lignes de programme, pour qu'elles soient acceptées par le Basic :

— En MSX-BASIC, on peut entrer jusqu'à 255 caractères sur une même ligne.

— Une ligne peut contenir plusieurs instructions, si celles-ci sont séparées par des ":".

***Exemple*** :

```
20 CLS : PRINT "BONJOUR" : PRINT "MONSIEUR"
```

— Un numéro de ligne doit être un entier compris entre 0 et 65529. Pour faciliter la mise au point des programmes, il est conseillé de choisir un incrément systématique de 10 pour des numéros successifs.

— Les espaces ne sont pas significatifs, mais sont parfois utiles pour améliorer la lisibilité des programmes.

### 1.2.4. Les messages d'erreur

Lors de l'exécution d'un programme ou d'une commande en mode direct, il arrive parfois que l'instruction ne soit pas acceptée par l'interpréteur du langage. Dans ce cas, un message d'erreur est affiché à la position du curseur et le déroulement du programme est interrompu. Cette facilité s'avère très utile pour mettre au point un programme, les différents messages servant de guide pour la recherche des fautes.

Toute instruction mal écrite, toute faute de ponctuation ou simplement toute instruction impossible à exécuter provoquent une erreur. 35 messages d'erreur sont prévus en MSX-BASIC et répertoriés à l'appendice 2.

***Exemples*** :

— En mode direct :

```
PRINTE "BONJOUR"        [RETURN]

Syntax error
Ok
```

— En mode programme :

```
10 CLS                  [RETURN]
20 PRINT "BONJOUR"      [RETURN]
30 PRINTE "BONJOUR"     [RETURN]

RUN                     [RETURN]

Syntax error in 30
Ok
```

Comme on peut le constater dans le deuxième exemple, le système de traitement des erreurs renvoie également le numéro de ligne où le programme s'est interrompu (voir § 3.3.6).

## 1.3. LE CLAVIER MSX

Le clavier MSX est constitué de 73 touches organisées selon le standard américain QWERTY. Très complet, il est de type machine à écrire avec de vraies touches mécaniques ! Toutes les touches alphanumériques sont regroupées sur la partie centrale, tandis que les touches spécialisées sont disposées à gauche et à droite de ce pavé. Un groupe de cinq touches de fonctions, fréquentes sur les ordinateurs professionnels, vient compléter ce clavier auquel il ne manque pratiquement rien.

### 1.3.1. Le principe

Chaque fois qu'une touche est actionnée sur le clavier, un code correspondant au caractère ou à la fonction de cette touche est envoyé à l'ordinateur (en mode direct, le caractère est directement affiché à la



**Le clavier MSX**

position du curseur clignotant). Ce code appelé **Code ASCII** est un entier compris entre 0 et 255. L'ensemble des caractères correspondant à tous les codes pouvant être traités par le MSX-BASIC constitue le **jeu de caractères** de l'ordinateur (voir appendice 1).

En dehors des classiques SHIFT et CAPS permettant la sélection et le blocage en majuscules, il est nécessaire de pouvoir générer tous les codes par le clavier. Pour ce faire, quelques touches spécialisées autorisent l'accès à tous les symboles graphiques du jeu de caractères en les combinant avec les touches alphanumériques standard : C'est le cas des touches GRAPH, CODE et DEAD.

Un certain nombre de codes sont associés non pas à des caractères, mais à des fonctions spéciales du Basic. Ces codes, compris entre 0 et 32 peuvent être générés au clavier par l'intermédiaire de la touche CTRL. En général, ces codes appelés **codes de contrôle** s'utilisent avec l'**éditeur** du MSX-BASIC (voir § 1.4).

Enfin, 11 touches spécialisées contrôlent le déplacement du curseur, l'accès rapide à des fonctions d'édition (HOME, INS, DEL, TAB), et l'interruption temporaire ou définitive durant l'exécution d'un programme (voir § 1.4).

### 1.3.2. Les touches de fonction

Les cinq touches de fonction méritent une attention particulière : Chacune d'entre elles est associée à deux commandes du Basic diffé-

rentes suivant que l'on soit en majuscules ou en minuscules. La syntaxe de ces instructions est rappelée en bas de l'écran, sur la 24e ligne.

Le principal intérêt de ces touches, est la possibilité de modifier par le Basic leur affectation. Le programmeur peut ainsi associer à ces cinq touches des fonctions spécifiques à son logiciel, avec un rappel de leur affectation toujours disponible à l'écran.

A la mise en route de l'ordinateur, les cinq touches mémorisent dix instructions courantes du Basic :

| MODE NORMAL | | | MODE MAJUSCULE | | |
|---|---|---|---|---|---|
| F1 | COLOR | (4.1.1) | F6 | COLOR 15,4,7 | (4.1.1) |
| F2 | AUTO | (3.3.1) | F7 | CLOAD" | (4.3) |
| F3 | GOTO | (3.3.3) | F8 | CONT | (3.3.1) |
| F4 | LIST | (3.3.1) | F9 | LIST | (3.3.1) |
| F5 | RUN | (3.3.1) | F10 | RUN | (3.3.1) |

## 1.4. ÉCRIRE UN PROGRAMME : L'ÉDITEUR

### 1.4.1. Généralités

Par le terme générique d' "éditeur d'écran", on entend l'ensemble des codes de contrôle associés à des touches du clavier servant à mettre au point un programme ou un texte en modifiant, déplaçant ou supprimant des caractères ou des lignes affichés à l'écran.

Dans le cas du MSX-BASIC, nous avons l'avantage de disposer d'un éditeur "pleine page", qui nous permet d'accéder directement à tout point d'une page à l'écran.

Ainsi, lorsqu'un programme est listé à l'écran, il est possible de positionner le curseur clignotant sur toutes les lignes visibles. Un certain nombre de touches ou de combinaisons de touches permettent ensuite d'agir sur le caractère ou la ligne où se trouve le curseur. Une fois les modifications effectuées, elles doivent être validées par l'intermédiaire de [RETURN].

— En mode direct, toute modification est directement prise en compte après appui sur la touche [RETURN].

— En mode programme, la ligne modifiée est mémorisée en lieu et place de celle qu'elle remplace.

### 1.4.2. Les touches réservées

Écrivez le petit programme suivant sur votre écran :

```
10 CLS
20 PRINT "JE SUIS UN EDITEUR PLEINE PAGE"
30 PRINT "LIVRE EN STANDARD SUR LE MSX"
40 PRINT "TRES SIMPLE A UTILISER"
```

— A droite et en bas du clavier de votre ordinateur, 4 touches disposées en losange indiquent les quatre bords de l'écran. Ces touches servent à déplacer le **curseur**. Celui-ci peut se mouvoir colonne par colonne dans toutes les directions, y compris en diagonale. Pour ce faire, il faut appuyer simultanément sur deux touches dans la direction souhaitée.

— Juste au dessus de ce pavé, quatre autres touches donnent accès à des fonctions plus complexes. Pour visualiser leur action, il suffit de se placer avec les touches curseur sur une des lignes du programme présent sur l'écran.

**HOME** renvoie le curseur en haut et à gauche de l'écran. En mode majuscules (combinée avec SHIFT), l'écran est effacé.

**BS** Efface le caractère sur lequel se trouve le curseur, et décale la ligne restante vers la gauche pour combler l'espace ainsi créé.

**DEL** Efface le caractère sur lequel se trouve le curseur et ramène la ligne restante à gauche.

**INS** Les caractères entrés après la frappe de cette touche sont insérés à partir du curseur. Cette commande est annulée si le curseur est déplacé à l'aide des flèches.

**TAB** Déplace le curseur vers la droite, à la position de tabulation suivante (une position toute les 14 colonnes).

Après modification d'un programme, la commande Basic **LIST** permet de vérifier que les changements ont été sauvegardés (voir § 3.3.1). Pour sortir d'une ligne sans valider les transformations, il suffit de presser simultanément la touche **CTRL** et la touche **STOP**.

### 1.4.3. Les codes de contrôle

Comme il a déjà été dit dans le chapitre 1.3 consacré au clavier, certains codes dont la valeur est inférieure à 32 ne correspondent pas à des caractères mais à des fonctions spéciales du Basic.

On peut utiliser ces codes de contrôle dans les programmes (voir CHR$ § 3.4.3), ou directement par le clavier, en combinant l'action des

touches alphanumériques avec la touche **CTRL**. Certains d'entre eux sont très utiles lors de la mise au point de programme par l'éditeur. Voici la liste complète de ces codes associés aux commandes équivalentes du clavier:

*Note*: Toutes les touches réservées de l'éditeur correspondent à un code de contrôle. Elles sont rappelées dans ce tableau à la rubrique FONCTION.

| | | |
|---|---|---|
| 1 | CTRL/A | NEANT. |
| 2 | CTRL/B | Place le curseur au début du mot suivant. |
| 3 | CTRL/C | Interrompt l'exécution du programme. |
| 4 | CTRL/D | NEANT. |
| 5 | CTRL/E | Efface la ligne vers la droite à partir du curseur. |
| 6 | CTRL/F | Place le curseur sur le début du mot suivant. |
| 7 | CTRL/G | Emet une sonnerie courte. |
| 8 | CTRL/H | Equivalent à **BS**. |
| 9 | CTRL/I | Equivalent à **TAB**. |
| 10 | CTRL/J | Descend le curseur d'une ligne. |
| 11 | CTRL/K | Equivalent à **HOME**. |
| 12 | CTRL/L | Efface l'écran (Equivalent à **CLS**). |
| 13 | CTRL/M | Equivalent à **RETURN**: Valide la ligne et place le curseur au début de la ligne suivante. |
| 14 | CTRL/N | Permet d'ajouter du texte à la ligne courante (Le curseur est automatiquement placé en fin de ligne). |
| 15 | CTRL/O | NEANT. |
| 16 | CTRL/P | NEANT. |
| 17 | CTRL/Q | NEANT. |
| 18 | CTRL/R | Equivalent à **INS**. |
| 19 | CTRL/S | NEANT. |
| 20 | CTRL/T | NEANT. |

| | | |
|---|---|---|
| 21 | CTRL/U | Efface la ligne où se trouve le curseur. |
| 22 | CTRL/V | NEANT. |
| 23 | CTRL/W | NEANT. |
| 24 | CTRL/X | Equivalent à **SELECT**: Pas d'action. |
| 25 | CTRL/Y | NEANT. |
| 26 | CTRL/Z | NEANT. |
| 27 | CTRL/[ | Equivalent à **ESC**: Pas d'action. |
| 28 | CTRL/¥ | Equivalent à **FLECHE DROITE**. |
| 29 | CTRL/] | Equivalent à **FLECHE GAUCHE**. |
| 30 | CTRL/^ | Equivalent à **FLECHE HAUT**. |
| 31 | CTRL/_ | Equivalent à **FLECHE BAS**. |
| 128 | NEANT | Equivalent à la touche **DEL**. |

# 2. Apprendre la programmation

Avant d'aborder les spécifications complètes du MSX-BASIC, il est indispensable de posséder quelques notions de programmation. Ce chapitre est destiné à guider le lecteur lors de ses premiers pas, à l'aide de quelques instructions parmi les plus simples du langage. Dans tous les cas, la syntaxe complète de ces commandes est omise, puisque présentée dans la deuxième partie de l'ouvrage.

*Note* : Il est conseillé au lecteur de se familiariser avec les commandes de l'éditeur du MSX avant d'aborder cette section (voir § 1.4). Rappelons qu'un programme se visualise avec **LIST** et s'exécute avec **RUN**.

## 2.1. LES VARIABLES

Pour pouvoir effectuer des calculs successifs et en mémoriser les résultats, il est indispensable de disposer de "tiroirs" où ranger des valeurs durant le déroulement d'un programme. En Basic, on appelle ces tiroirs des **variables**. On peut stocker dans des variables des nombres, des caractères ou même des groupes de données.

Pour caractériser une variable, on lui donne un nom, qui peut être constitué d'un ou plusieurs caractères. L'instruction du Basic qui permet d'affecter une valeur à un nom s'appelle **LET**:

LET VALEUR=50 suivi de [RETURN] affecte le nombre 50 à la variable dont l'identificateur est VALEUR.

En pratique, cela signifie que l'on peut remplacer chaque fois que cela est nécessaire le chiffre 50 par le mot VALEUR.

Dans un programme, l'affectation n'est effectuée que lors de l'exécution:

```
10 LET VALEUR=50        [RETURN]
20 LET NOMBRE=12        [RETURN]
30 LET LIGNE=30         [RETURN]
```

Après écriture de ce programme, les trois variables ne sont pas assignées et valent 0. Il faut lancer l'exécution avec RUN pour réaliser cette assignation.

REMARQUE: Toute variable numérique non assignée a pour valeur par défaut 0.

Il faut noter que l'instruction LET n'est pas obligatoire, et peut donc être omise lors de l'écriture des programmes: Dans l'exemple qui précède, on écrira VALEUR=50 sans se soucier de LET.

Trois types principaux de variables sont reconnus par les interpréteurs Basicş:

— Les variables numériques, qui peuvent être entières, en simple et en double précision Dans ce chapitre, nous n'étudierons que le cas des variables numériques en double précision, ce qui correspond à la condition par défaut.

— Les variables chaînes.

— Les tableaux.

### 2.1.1. Les variables numériques

On appelle variables numériques toutes les variables destinées à stocker des valeurs de type nombre entier ou réél, positif ou négatif. On peut effectuer sur ces variables toutes les opérations arithmétiques courantes, telles que addition, multiplication, soustraction, etc... (voir liste complète au § 3.2.2). Les règles de syntaxe sont identiques à celles pratiquées habituellement en arithmétique, seuls les symboles utilisés diffèrent :

```
10 VALEUR1 = 50
20 VALEUR2 = 2
30 SOMME = VALEUR1 + VALEUR2               Addition
40 SOUST = VALEUR1 - VALEUR2               Soustraction
50 PRODUIT = VALEUR1 * VALEUR2             Multiplication
60 DIVISION = VALEUR1 / VALEUR2            Division
```

Il est possible de combiner plusieurs opérations entre elles, et d'utiliser des parenthèses pour fixer un ordre de priorité :

```
VALEUR = (5+2)*3
```

Les parenthèses peuvent être imbriquées sur plusieurs niveaux :

```
VALEUR = (5+(8/2))-1
```

REMARQUES :

— Sans parenthèses, les multiplications et les divisions sont effectuées avant les additions et les soustractions.

— Les espaces ne sont pas significatifs et peuvent être utilisés pour améliorer la présentation d'un programme.

### 2.1.2. Les variables chaînes

Le Basic autorise la manipulation de chaînes de caractères composées d'une suite de caractères alphanumériques. Une variable chaîne suit des règles différentes de celles des variables numériques :

— Le nom d'une variable chaîne doit être constitué d'une suite de caractères terminée par le symbole "$" :

```
A$, VAR$, FONCTION$, JEU1$ sont des noms autorisés.
```

REMARQUE : Pour tous les types de variables le nom ne doit pas commencer par un chiffre.

— Toutes chaîne de caractères doit être limitée par des guillemets pour être reconnue par le Basic :

```
"JE SUIS UNE CHAINE DE CARACTERES"
```

255 représente le nombre 255
"255" représente une suite de trois caractères.

— L'affectation d'une chaîne à une variable du même type suit des règles identiques pour les chaînes et les nombres :

```
LET TIROIR$ = "CHAINE DE CARACTERE"
```

REMARQUE : La chaîne entre guillemets ne peut pas dépasser 255 caractères.

— Tous les caractères sont acceptés à l'intérieur d'une chaîne, à l'exception des guillemets, qui sont reconnus par l'interpréteur Basic en tant que séparateurs pour les instructions d'affichage (voir § 2.2).

— Il est possible d'effectuer des opérations et des comparaisons sur des chaînes de caractères. L'opération la plus courante est appelée **concaténation**, elle sert à relier entre elles plusieurs chaînes. Pour effectuer une concaténation, on procède comme pour une addition qu'il s'agisse d'une variable ou d'une constante :

```
LET DEBUT$="JE SUIS "
LET MILIEU$="UNE CHAINE "
LET FIN$="DE CARACTERES "

LET SOMME$=DEBUT$+MILIEU$+FIN$
```

SOMME$ contiendra la phrase résultat de la concaténation des trois variables. On aurait pu aussi écrire :

```
SOMME$= "JE SUIS "+"UNE CHAINE "+"DE CARACTERES"
```

REMARQUE : A l'intérieur d'une chaîne de caractères, les espaces sont significatifs et sont traités comme des caractères.

### 2.1.3. Les tableaux

#### Tableaux à une dimension

Lorsqu'on dispose d'un grand nombre de variables à traiter, il est possible de les organiser en tableaux, avec un seul nom pour les carac-

tériser et un indice pour les repérer. Prenons un exemple simple :

On dispose de cinq valeurs correspondant au cours moyen d'une action depuis cinq ans. On peut écrire simplement :

```
10 COURS78 = 10000
20 COURS79 = 11000
30 COURS80 = 12000
40 COURS81 = 10000
50 COURS82 = 9000
```

En utilisant un tableau numérique, le programme devient :

```
10 COURS(1) = 10000
20 COURS(2) = 11000
30 COURS(3) = 12000
40 COURS(4) = 10000
50 COURS(5) = 9000
```

Un seul nom de variable, COURS, est utilisé pour caractériser le tableau, et ce quelque soit sa taille. On accède à chaque élément en modifiant la valeur de l'indice entre parenthèses. Les variables composant un tableau sont appelées **variables indicées**.

Dans l'exemple qui précède, le tableau est constitué de cinq cases simples. Jusqu'à dix cases, on peut directement organiser des variables en tableau sans effectuer de déclaration. Il est également possible de définir des tableaux avec un nombre plus important de cases, mais ils doivent dans ce cas être déclarés par une instruction spécialisée, **DIM** (voir § 3.3.5).

REMARQUES :

— Le premier élément utilisable d'un tableau a pour numéro 0. On peut donc écrire :

```
10 COURS(0) = 10000
20 COURS(1) = 11000
30 COURS(2) = 12000
40 COURS(3) = 10000
50 COURS(4) = 9000
```

— Avant utilisation, tous les éléments d'un tableau numérique sont initialisés à 0.

L'emploi de tableau est recommandé chaque fois qu'un grand nombre d'informations relatives à un même sujet doit être traités. Tous les types de variables peuvent être constitués en tableaux :

```
10 NOM$(0) = "JEAN"
20 NOM$(1) = "PIERRE"
30 NOM$(2) = "LOUIS"
40 NOM$(3) = "GEORGES"
```

NOM$( ) est un tableau de chaînes regroupant quatre constantes de chaînes. Chaque élément du tableau peut contenir une chaîne dont la longueur maximum est 255 caractères. Dans le cas des tableaux de chaînes, le contenu d'une case à l'initialisation correspond à une chaîne vide.

**Tableaux à plusieurs dimensions**

Il est également possible de travailler sur des tableaux à plusieurs dimensions. Reprenons notre exemple numérique :

On voudrait étendre le calcul boursier au cours de trois actions différentes sur cinq années. Avec des variables simples, il est nécessaire de définir 15 noms différents, un pour chaque action. En ajoutant une dimension au tableau, on constitue un damier de 3 × 5 cases contenant toutes les valeurs.

```
10 COURS(1,1) = 10000:COURS(1,2) = 11000:COURS(1,3) = 2000
20 COURS(2,1) = 11000:COURS(2,2) = 12000:COURS(2,3) = 1500
30 COURS(3,1) = 12000:COURS(3,2) = 13000:COURS(3,3) = 1200
40 COURS(4,1) = 10000:COURS(4,2) = 17000:COURS(4,3) = 1000
50 COURS(5,1) = 9000 :COURS(5,2) = 16000:COURS(5,3) = 1100
```

Le premier indice du tableau permet d'accéder aux cinq années étudiées, tandis que le deuxième indice différencie les trois actions (chacun des indices est lié à une dimension du tableau). Un tableau à deux dimensions peut être comparé à un livre dont le numéro de page correspond au premier indice et le numéro de ligne dans la page au deuxième indice.

REMARQUES :

— Les indices d'un tableau peuvent être des constantes ou des variables.

— Le premier indice utilisable a pour numéro 0.

— Des variables de types différents peuvent avoir le même nom : Le MSX-BASIC ne confondra pas COURS, COURS$, COURS(1,4).

— Le nombre de dimensions n'est pas limité, mais un chiffre supérieur à trois est rarement employé pour des raisons pratiques.

— Quelque soit le nombre de dimensions du tableau, chacune d'elles ne peut contenir que dix éléments à moins de passer par l'instruction **DIM** (voir § 3.3.5).

## 2.2. AFFICHER SUR L'ÉCRAN

Pour pouvoir visualiser des données ou les résultats d'un calcul, il faut disposer d'une instruction permettant d'afficher à l'écran. Les spécifications du MSX-BASIC incluent un grand nombre de commandes réalisant des affichages selon des formats différents, parmi lesquelles **PRINT** est certainement la plus simple à utiliser : On peut par son intermédiaire afficher des valeurs numériques, des variables, des chaînes de caractères (voir aussi § 3.3.2).

— Pour afficher une constante, il suffit de faire suivre l'instruction de la valeur à afficher, sans inclure de séparateurs :

`PRINT 200` Affiche le nombre 200 à la position du curseur.
`PRINT 55*3` Affiche le résultat du calcul.

— Pour afficher une chaîne de caractère, on doit conserver les guillemets de part et d'autre du texte à imprimer :

```
PRINT "JE SUIS UNE CHAINE DE CARACTERES"
```

— Pour imprimer des variables, on utilise directement leur identificateur sans séparateur :

```
10 VALEUR = 50          [RETURN]
20 PRINT VALEUR         [RETURN]
```

***Exemple*** :

```
10 VALEUR1 = 50                          [RETURN]
20 VALEUR2 = 100                         [RETURN]
30 SOMME = VALEUR1 + VALEUR2             [RETURN]
40 TEXTE$="RESULTAT "                    [RETURN]
50 PRINT TEXTE$                          [RETURN]
60 PRINT SOMME                           [RETURN]

RUN [RETURN]
```

Après exécution, l'écran affichera :

```
RESULTAT
150
```

PRINT pouvant directement afficher le résultat d'un calcul, on peut supprimer la ligne 30 et remplacer la ligne 60 par :

```
60 PRINT VALEUR1 + VALEUR2
```

Les parenthèses sont bien sûr autorisées pour imposer l'ordre de priorité du calcul :

```
60 PRINT (VALEUR1 + VALEUR2)*3
```

Pour les chaînes de caractères, la concaténation peut être effectuée sur la même ligne que l'impression :

```
50 PRINT "RESULTAT " + "DU CALCUL :"

50 PRINT TEXTE$ + "DU CALCUL :"
```

sont des syntaxes correctes.

REMARQUE : Il ne faut pas confondre une variable numérique et la constante de chaîne qui lui correspond :

| | |
|---|---|
| `PRINT VALEUR1` | Affiche le contenu de la variable. |
| `PRINT "VALEUR1"` | Affiche le texte inclus dans les guillemets. |

**Les séparateurs**

Chaque fois qu'un PRINT est exécuté, le message qui le suit est affiché à la position du curseur clignotant, puis un passage à la ligne est effectué. Ce saut de ligne équivaut à un appui sur [RETURN] en mode programme. Si aucun argument n'est associé à PRINT, seul le saut de ligne est effectué. On utilise cette possibilité pour améliorer la présentation des messages à l'écran.

```
10 PRINT "AFFICHAGE DES CALCULS :"
20 PRINT
30 PRINT 22*5
40 PRINT
50 PRINT 18*4
```

Après exécution, les messages seront séparés par des lignes vides :

```
AFFICHAGE DES RESULTATS :

110

72
```

Il est possible d'inhiber ce saut de ligne à l'aide de deux séparateurs, la virgule et le point-virgule. Placés sur la même ligne que le PRINT à la suite de l'argument, ces deux caractères imposent la position du curseur et donc l'endroit où aura lieu le prochain affichage.

— Dans le cas du point-virgule, le curseur se place à la suite du dernier caractères imprimé :

```
10 PRINT "AFFICHAGE DES CALCULS : ";
20 PRINT 22*5;
30 PRINT " ";
40 PRINT 18*4
```

Donnera sur l'écran :

**AFFICHAGE DES CALCULS :  110  72**

— La virgule fonctionne comme un tabulateur, elle autorise un espacement par groupe de huit caractères sur une même ligne :

```
10 PRINT "AFFICHAGE DES CALCULS :",
20 PRINT 22*5,
30 PRINT 18*4
```

Donnera sur l'écran :

**AFFICHAGE DES CALCULS :       110      72**

REMARQUE : Plusieurs ordres d'affichages peuvent être groupés dans une seule instruction PRINT. Les séparateurs permettent de les délimiter entre eux et de définir leur espacement. Le programme qui précède peut donc s'écrire :

```
10 PRINT "AFFICHAGE DES CALCULS :",22*5,18*4
```

Enfin, plusieurs types de variables et de séparateurs peuvent être associés sur une même ligne pour rendre l'écriture plus compacte :

```
10 VALEUR1 = 100
20 VALEUR2 = 50
30 TEXTE1$="SOMME = "
50 TEXTE2$="DIFFERENCE = "
60 PRINT TEXTE$;VALEUR1+VALEUR2,TEXTE2$;VALEUR1-VALEUR2
```

REMARQUES :

— Une instruction spécialisée du MSX-BASIC permet de placer la position d'écriture à n'importe quel endroit de l'écran en spécifiant un couple de coordonnées (voir LOCATE § 4.1.1).

— PRINT peut être remplacé par le caractère "?" pour accélérer l'écriture d'un programme.

## 2.3. SAISIR DES DONNÉES AU CLAVIER

Les variables servent à mémoriser des calculs, PRINT à les afficher. Pour établir un dialogue avec l'ordinateur, il est maintenant nécessaire de saisir des données fournies par l'utilisateur durant le déroulement d'un programme. L'instruction INPUT permet d'affecter à une variable une entrée clavier :

```
10 INPUT VALEUR
20 PRINT
30 PRINT "ENTREE :";VALEUR
```

Après la frappe de RUN suivie de [RETURN], un point d'interrogation est affiché à la position courante d'écriture et le curseur clignotant est affiché à la droite de ce caractère. Cet affichage indique que la commande INPUT attend l'entrée d'une valeur numérique au clavier. Une fois le nombre entré, la frappe de [RETURN] valide l'entrée, la variable VALEUR est initialisée et le déroulement du programme se poursuit.

```
? _
? 25        [RETURN]
ENTREE : 25
```

REMARQUE : Si l'on entre [RETURN] sans aucune valeur numérique, VALEUR est égale à 0.

Tout type de variable peut être associé à INPUT, qui peut servir à saisir des chaînes de caractères :

```
10 INPUT NOM$
20 PRINT
30 PRINT "BONJOUR ";NOM$
```

Donnera sur l'écran :

```
? _
? NICOLAS             [RETURN]
BONJOUR NICOLAS
```

REMARQUES :

— La chaîne de caractères doit être entrée sans guillemets. Si ce caractère est inclus dans la chaîne, il sera traité comme une lettre et saisi. Le séparateur "," n'est pas accepté.

— Les données saisies au clavier doivent correspondre au type de la variable. Dans le cas contraire, un message d'erreur est affiché et le point d'interrogation indique à nouveau l'attente d'une entrée au clavier.

```
10 INPUT VALEUR
20 PRINT VALEUR
```

```
RUN              [RETURN]
```

```
? NICOLAS        [RETURN]
```
Une chaîne de caractères est entrée

```
? Redo from start
?_
```
L'ordinateur veut une valeur numérique !

La confusion est toujours possible, c'est pourquoi l'instruction INPUT prévoit l'inclusion d'un message complet qui s'affiche dans ce cas à la place du point d'interrogation. Ce message doit précéder le nom de la variable, être limité par des guillemets, et suivi d'un point-virgule. Son emploi clarifie les programmes et permet la mise au point d'une conversation complète avec l'ordinateur.

```
10 INPUT "QUEL EST VOTRE NOM ";NOM$
20 PRINT
30 INPUT "QUEL EST VOTRE AGE ";AGE
40 PRINT
50 PRINT "VOUS VOUS APPELEZ ";NOM$;
60 PRINT "ET VOUS AVEZ ";AGE;" ANS!"
```

Donnera sur l'écran :

```
QUEL EST VOTRE NOM ? NICOLAS   [RETURN]

QUEL EST VOTRE AGE ? 30        [RETURN]

VOUS VOUS APPELEZ NICOLAS ET VOUS AVEZ 30 ANS!
```

— INPUT peut combiner plusieurs entrées sur une même ligne. Dans ce cas, chaque nom de variable doit être suivi d'une virgule. Lors de l'entrée des données, chacune d'elles doit également être séparée de la suivante par une virgule. Cette facilité d'écriture permet de condenser les programmes. L'exemple précédent peut s'écrire :

```
10 INPUT "QUEL EST VOTRE NOM, VOTRE AGE ";NOM$,AGE
20 PRINT
50 PRINT "VOUS VOUS APPELEZ ";NOM$;
60 PRINT "ET VOUS AVEZ ";AGE;" ANS!"
```

Ce qui donnera sur l'écran :

**QUEL EST VOTRE NOM, VOTRE AGE ?** NICOLAS,30 [RETURN]

**VOUS VOUS APPELEZ NICOLAS ET VOUS AVEZ 30 ANS!**

Plusieurs types de variables peuvent coexister sur une même ligne de saisie, mais il est indispensable de rentrer les données en conservant l'ordre d'écriture pour que les types correspondent.

## 2.4. LES BRANCHEMENTS

Lors du déroulement d'un programme, les instructions sont exécutées dans l'ordre croissant des numéros de lignes, et ce jusqu'à la dernière instruction de la dernière ligne. Ce mode de fonctionnement linéaire ne correspond pas aux conditions réelles de la programmation. Il est en effet indispensable de pouvoir "aiguiller" le programme vers une autre section sans perturber son fonctionnement. GOTO est la commande la plus simple permettant de réaliser ce branchement :

```
10 PRINT "BONJOUR ";
20 GOTO 40
30 PRINT "MONSIEUR ";
40 PRINT "MADAME ";
```

Donnera **BONJOUR MADAME** à l'écran :

Dans l'exemple qui précède, le branchement réalisé par GOTO permet de sauter une des lignes du programme. L'argument de GOTO, est le numéro de la ligne à partir de laquelle l'exécution doit se poursuivre.

La notion de branchement est étroitement liée à celle des boucles : il est très simple d'employer GOTO pour exécuter en boucle une même section d'un programme :

```
10 INPUT "QUEL EST VOTRE NOM, VOTRE AGE ";NOM$,AGE
20 PRINT
30 PRINT "VOUS VOUS APPELEZ ";NOM$;
40 PRINT "ET VOUS AVEZ ";AGE;" ANS!"
50 PRINT
60 GOTO 10
```

Cette boucle élémentaire présente l'inconvénient d'être infinie ! Une fois les données saisies, elles sont affichées et la saisie est à nouveau effectuée. Nous verrons dans le chapitre suivant comment fixer une condition permettant la sortie d'une boucle de ce type. En attendant, **CTRL/STOP** vous permettra de sortir de cette situation délicate...

## 2.5. LES INSTRUCTIONS CONDITIONNELLES

On a vu comment on pouvait bâtir une "conversation" avec un ordinateur par l'intermédiaire d'INPUT, et modifier la séquence d'exécution d'un programme à l'aide de GOTO. Comment faire pour lier ces deux instructions, c'est-à-dire décider d'exécuter telle ligne de programme plutôt qu'une autre en fonction des données entrées par l'utilisateur ? Les instructions IF et THEN servent à tester une condition et à exploiter le résultat de ce test :

**IF** condition **THEN** instruction

La condition la plus simple peut être l'égalité avec un nombre quelconque.

```
10 INPUT "DONNEZ LA VALEUR DE A";A
20 IF A=10 THEN PRINT "A EST EGAL A 10"
```

Dans cet exemple, le message ne s'affiche que si la **condition** A=10 est vraie. L'**instruction** associée PRINT peut se remplacer par n'importe quelle commande du Basic. Le test peut également fonctionner avec des chaines de caractères.

```
10 INPUT "QUEL EST VOTRE SEXE (M/F) ";S$
20 PRINT "BONJOUR ";
30 IF S$="M" THEN PRINT "MONSIEUR"
40 IF S$="F" THEN PRINT "MADAME"
```

Dans ce cas, les lignes 30 et 40 affichant un message en fonction de la réponse de l'utilisateur !

Les **Opérateurs relationnels** et **logiques** peuvent être associés afin de sophistiquer le traitement d'une condition (voir définition § 1.1.3):

```
10 INPUT "DONNEZ LA VALEUR DE A";A
20 IF A>0 AND A<10 THEN PRINT "A EST COMPRIS ENTRE 0 ET 10"
```

Ce qui peut se traduire par : **SI** A est supérieur à 0 **ET** A est inférieur à 10 **ALORS**...

Pour contrôler plus efficacement le déroulement du programme, les instructions conditionnelles IF et THEN associées à l'instruction GOTO permettent de réaliser des **branchements conditionnels**:

```
10 INPUT "QUEL EST VOTRE SEXE (M/F) ";S$
20 IF S$<>"M" AND S$<>"F" THEN GOTO 10
30 PRINT "BONJOUR ";
40 IF S$="M" THEN PRINT "MONSIEUR"
50 IF S$="F" THEN PRINT "MADAME"
```

Si le résultat de la saisie n'est pas l'un des deux caractères demandés, la ligne 20 aiguille le programme sur la ligne 10. Dans le cas contraire, les lignes 30 et 40 affichent le message adéquat. Il est maintenant possible de contrôler la boucle infinie du paragraphe 2.4:

```
10 INPUT "QUEL EST VOTRE NOM, VOTRE AGE ";NOM$,AGE
20 PRINT
30 PRINT "VOUS VOUS APPELEZ ";NOM$;
40 PRINT "ET VOUS AVEZ ";AGE;" ANS!"
50 PRINT
60 IF NOM$<>"" THEN GOTO 10
70 PRINT "TERMINE, MERCI"
```

Tant que des noms sont entrés dans le programme, la saisie et les affichages continuent. Pour interrompre le déroulement, il suffit de répondre [RETURN] à la question de la ligne 10. Aucun caractère n'ayant été saisi, la chaîne NOM$ est vide, et le programme se termine.

***Exemple***: Nous en savons maintenant assez pour créer un petit programme de "conversation assistée par ordinateur", très utile dans les moments de solitude!

```
10 CLS
20 INPUT "QUEL EST VOTRE NOM ";NOM$
30 PRINT
40 PRINT "BONJOUR ";NOM$:PRINT "COMMENT ALLEZ-VOUS, BIEN OU MAL
   ";
50 INPUT A$
60 IF A$="BIEN" THEN GOTO 120
70 IF A$="MAL" THEN GOTO 150
80 PRINT
90 PRINT "SI VOUS NE REPONDEZ PAS CORRECTEMENT,":PRINT "COMMENT
   VOULEZ VOUS QUE JE FASSE MON":PRINT "TRAVAIL ? "
100 PRINT:INPUT "ALORS, CA VA BIEN OU MAL ";A$
110 GOTO 60
120 PRINT: PRINT "SI CA VA SI BIEN QUE CA, C'ETAIT":PRINT "PAS
    LA PEINE DE ME DERANGER"

130 PRINT: PRINT"SI VOUS AVEZ BESOIN DE MOI, N'HESITEZ":PRINT
    "PAS A ME PASSER UN COUP DE FIL"
140 GOTO 610
```

```
150 PRINT :PRINT "JE SUIS PROGRAMME POUR RESOUDRE TOUTES":PRINT
    "SORTES DE PROBLEMES, SAUF CEUX PORTANT":PRINT "SUR LA
    REPRODUCTION DES MOLLUSQUES"
160 PRINT:PRINT "QUEL TYPE DE PROBLEME AVEZ-VOUS, ";NOM$
170 PRINT :INPUT"SEXE, ARGENT, TRAVAIL ";PRO$
180 IF PRO$="SEXE" THEN GOTO 250
190 IF PRO$="ARGENT" THEN GOTO 400
200 IF PRO$="TRAVAIL" THEN GOTO 500
210 PRINT:PRINT"N'AYEZ PAS PEUR DE REPONDRE FRANCHEMENT"
220 PRINT NOM$;" NOUS SOMMES ENTRE NOUS"
230 PRINT:PRINT "ALORS, QUELS PROBLEMES ?"
240 GOTO 170
250 CLS
260 PRINT:PRINT"VOTRE PROBLEME, C'EST TROP OU PAS ASSEZ";
270 INPUT S$
280 IF S$="TROP" THEN GOTO 320
290 IF S$="PAS ASSEZ" THEN GOTO 340
300 PRINT:PRINT"IL EST DES SUJETS AVEC LESQUELS ON NE ":PRINT
    "PLAISANTE PAS ";NOM$:PRINT:PRINT "ALORS, TROP OU PAS
    ASSEZ";
310 GOTO 270
320 PRINT :PRINT "FRANCHEMENT, ";NOM$:PRINT "VOUS APPELEZ CA UN
    PROBLEME!"
330 GOTO 640
340 PRINT:PRINT"TOUT PEUT S'ARRANGER ";NOM$
350 PRINT"QUE FAITES VOUS SAMEDI SOIR ";
360 INPUT B$
370 PRINT :PRINT "C'EST BIEN CE QUE JE PENSAIS, VOUS":PRINT
    "ETES UNE SORTE DE MALADE"
380 PRINT "IL VAUT MIEUX CHANGER DE SUJET":
390 GOTO 640
400 CLS
410 PRINT :PRINT "C'EST ENNUYEUX, J'AI MOI-MEME":PRINT "DE
    GROSSES DIFFICULTES EN CE MOMENT"
420 PRINT :INPUT "POURREZ VOUS PAYER MES HONORAIRES";R$
430 IF R$="NON" THEN GOTO 610
440 IF R$="OUI" THEN GOTO 480
450 PRINT "NE TOURNEZ PAS AUTOUR DU POT ";NOM$
460 INPUT"ALORS OUI OU NON ";R$
470 GOTO 430
480 PRINT :PRINT "C'EST PAS SYMPA DE MENTIR ";NOM$
490 GOTO 640
500 CLS
510 PRINT "JE VOUS COMPREND, BIEN QUE JE SOIS":PRINT "IMMUNISE
    CONTRE CETTE ETRANGE VIRUS"
520 PRINT :PRINT "MON CONSEIL: CESSEZ IMMEDIATEMENT TOUTE"
530 PRINT "ACTIVITE QUELQU'ELLE SOIT"
540 PRINT :PRINT "VOULEZ VOUS QUE JE RESOLVE LES":PRINT
    "PROBLEMES D'ARGENT QUE CE CONSEIL"
550 PRINT "VA DECLENCHER IRREMEDIABLEMENT ";
560 INPUT R$
570 IF R$="OUI" THEN GOTO 400
580 IF R$="NON" THEN GOTO 640
590 PRINT :INPUT "HEIN ";R$
600 GOTO 570
610 PRINT :PRINT "CONTENT DE VOUS AVOIR CONNU ";NOM$
620 PRINT:PRINT "AU REVOIR, A LA PROCHAINE!"
630 END
640 PRINT :INPUT"D'AUTRES PROBLEMES ";R$
```

```
650 IF R$="OUI" THEN GOTO 160
660 IF R$="NON" THEN GOTO 610
670 PRINT:PRINT "VOTRE JARGON M'EST INCONNU ";NOM$
680 INPUT"REPONDEZ PAR OUI OU PAR NON ";R$
690 GOTO 650
```

REMARQUES :

— Il est possible de regrouper plusieurs instructions sur une même ligne en les séparant par ":".

— L'instruction **END** de la ligne 630 sert à terminer le programme, l'instruction **CLS** à effacer l'écran.

— Le chapitre 2.7 montre comment l'emploi de sous-programmes permet de remplacer les GOTO, et de structurer la programmation.

## 2.6. LES BOUCLES

On a vu aux paragraphes qui précèdent comment répéter un grand nombre de fois la même opération dans un programme en reprenant constamment l'exécution au même point avec GOTO. Ce type de boucles, très simple à mettre en œuvre n'autorise pas le contrôle du nombre de "tours" effectués par la boucle. Pour ce faire, il faut ajouter un **compteur** dans la boucle, et tester sa valeur à chaque itération :

```
10 LET COMPTE = 0
20 INPUT "QUEL EST VOTRE NOM ";NOM$
30 PRINT
40 PRINT "VOUS VOUS APPELEZ ";NOM$
50 PRINT
60 LET COMPTE = COMPTE + 1
70 IF COMPTE <= 20 THEN GOTO 20
80 PRINT "TERMINE"
```

La variable COMPTE est incrémentée à chaque passage par la ligne 60. Une fois la valeur 21 atteinte, la condition de la ligne 70 n'est plus remplie et le programme se termine.

Cette méthode un peu lourde à utiliser peut être remplacée par un groupe d'instructions plus adaptées à la gestion des boucles :

```
10 FOR COMPTEUR=[valeur initiale] TO [valeur finale]

20..50 INSERER LE PROGRAMME

60 NEXT COMPTEUR
```

Les instructions **FOR** et **TO** permettent le contrôle du nombre des répétitions, et l'instruction **NEXT** redémarre l'exécution à la première des lignes incluses dans la boucle jusqu'à la fin du traitement. A chaque itération, la variable compteur est incrémentée de 1.

Reprenons le programme précédent :

```
10 FOR COMPTE = 0 TO 20
20 INPUT "QUEL EST VOTRE NOM ";NOM$
30 PRINT
40 PRINT "VOUS VOUS APPELEZ ";NOM$
50 PRINT
60 NEXT COMPTE
70 PRINT "TERMINE"
```

```
FOR COMPTE=0     correspond à             LET COMPTE=0

TO 20            correspond au test       IF  COMPTE <= 20

NEXT COMPTE      remplace à la fois       COMPTE = COMPTE + 1
                                          GOTO 20
```

REMARQUES :

— [valeur finale] peut être une constante ou une variable. On aurait pu écrire :

```
10 FIN = 20
20 FOR COMPTE = 0 TO FIN
...
```

— En sortie de boucle, la variable COMPTE est égale à [valeur finale]+1.

— On peut imbriquer les boucles entre elles, et choisir pour l'incrément une autre valeur que 1 (voir détails § 3.3.3).

## 2.7. STRUCTURER UN PROGRAMME

Il est déjà possible de rédiger des petits programmes en se servant uniquement des instructions présentées dans cette section. L'emploi systématique de GOTO présente toutefois l'inconvénient d'embrouiller considérablement les listings lors d'aiguillages nombreux et répétés. Plusieurs méthodes sont disponibles pour améliorer la structure des programmes :

— Dans le cas où la même séquence d'instructions doit être utilisée plusieurs fois, le Basic offre la possibilité de constituer des **sous-programmes** permettant d'éviter la réécriture de sections identiques. Deux instructions de branchement sont disponibles:

**GOSUB** dont l'emploi est identique à celui de GOTO.

**RETURN** qui reprend l'exécution à la ligne suivant le dernier GOSUB rencontré.

Chaque fois qu'un **GOSUB** [N° de ligne] est rencontré dans un programme, le déroulement se poursuit au numéro de ligne spécifié, jusqu'à exécution d'un **RETURN**. La section de programme comprise entre [N° de ligne] et **RETURN** constitue un **sous-programme** qui peut être appelé autant de fois que cela est nécessaire à partir de n'importe quel point du programme principal.

***Exemple***:

```
10 CLS
20 PRINT "QUEL EST VOTRE NOM ";
30 INPUT R$
40 PRINT "QUEL EST VOTRE PRENOM ";
50 INPUT S$
60 PRINT "QUEL EST VOTRE AGE ";
70 INPUT T$
80 PRINT "VOUS VOUS APPELEZ ";S$;" ";R$;" AGE :";T$;" ANS"
```

On peut regrouper les messages dans un sous-programme démarrant à la ligne 100:

```
10 CLS
20 GOSUB 100
30 INPUT "NOM ";R$
40 GOSUB 100
50 INPUT "PRENOM ";S$
60 GOSUB 100
70 INPUT "AGE ";T$
80 PRINT "VOUS VOUS APPELEZ ";S$;" ";R$;" AGE :";T$;" ANS"
90 END
100 PRINT
110 PRINT "QUEL EST VOTRE ";
120 RETURN
```

— L'instruction **REM** sert à insérer des commentaires non exécutables à l'intérieur d'un programme. Tout caractère placé dans une ligne à droite d'une remarque est considérée comme commentaire, et ignoré par l'interpréteur Basic. L'emploi de REM simplifie la mise au point et facilite les améliorations ultérieures. Le programme précédent une fois commenté s'écrit:

```
10 REM EFFACEMENT DE L'ECRAN
20 CLS
30 REM SAISIE DU NOM
40 GOSUB 100
50 INPUT "NOM ";R$
60 REM SAISIE DU PRENOM
70 GOSUB 100
80 INPUT "PRENOM ";S$
90 REM SAISIE DE L'AGE
100 GOSUB 100
110 INPUT "AGE ";T$
120 REM AFFICHAGE DU RESULTAT
130 PRINT "VOUS VOUS APPELEZ ";S$;" ";R$;" AGE :";T$;" ANS"
135 END
140 REM SOUS PROGRAMME, MESSAGE DE SAISIE
150 PRINT
160 PRINT "QUEL EST VOTRE ";
170 RETURN
```

REMARQUES :

— REM est une instruction gourmande en mémoire, il faut donc l'utiliser avec parcimonie dans les très longs programmes.

— REM peut être remplacé par le caractère “'”

— Il est possible d'utiliser les remarques sur la fin d'une ligne contenant des instructions. Le contenu de la ligne n'est ignoré qu'à partir de la commande REM :

```
10 PRINT "BONJOUR": REM AFFICHE BONJOUR
```

*DEUXIÈME PARTIE*

# LE BASIC MSX, MANUEL DE RÉFÉRENCE

Cette section récapitule toutes les instructions et fonctions du MSX-BASIC. Certaines des informations qu'elle contient ont déjà été abordées de manière moins détaillée dans le petit cours de programmation de la première partie. Il s'agit ici d'un manuel de référence, destiné à toutes les machines respectant les spécifications du standard MSX.

# 3. Le Basic d'usage général

On appelle Basic d'usage général toutes les commandes du langage dont la fonction ne les destine pas à gérer les périphériques, le graphique ou le son. Cette présentation comprend une introduction récapitulant les spécifications du langage et les conventions d'écriture employées dans toute cette partie de l'ouvrage.

## 3.1. CONVENTIONS D'ÉCRITURE

Pour faciliter la compréhension, nous présenterons les instructions et fonctions de la manière suivante :

*Syntaxe* : Donne la syntaxe complète de l'instruction ou de la fonction.

*Application(s)* : Donne le détail de l'utilisation de la commande.

REMARQUE(S) : Dans cette rubrique seront exposées les particularités de la commande.

*Voir aussi* : Toutes les instructions ou fonctions du Basic liées à la commande sont regroupées sous cette rubrique.

***Exemple(s)*** : Selon la complexité des cas, un ou plusieurs exemples seront donnés :

— Tous les mots écrits en majuscules représentent les mots-clés du Basic, et sont à entrer tels quels au clavier.

— Les mots écrits en minuscules et compris entre // sont les données à fournir par le programmeur.

— Les données facultatives sont incluses entre crochets [].

— Toutes les ponctuations, hormis les deux précédentes, font partie intégrante de la syntaxe et doivent être tapées telles quelles.

— En ce qui concerne les nombres, nous adopterons les conventions suivantes :

X, Y et Z représentent des nombres en virgule flottante.

N, M des nombres entiers.

A représente une adresse mémoire (entier compris entre 0 et 65535).

***Exemple*** :

```
INPUT [/"commentaire";/] /variable 1/ [/,variable 2/]
```

— L'instruction INPUT est la commande Basic.

— Le commentaire est optionnel, sinon il doit être entre guillemets " " et suivi de ;.

— Au moins une variable doit être donnée comme argument.

— On peut si on le désire mentionner plusieurs variables, en les séparant par des virgules.

— Il est correct d'écrire :

INPUT "Donnez votre âge et votre mois de naissance";A,M$

## 3.2. SPÉCIFICATIONS DU LANGAGE

Avant d'aborder les types de données que peut manipuler le MSX-BASIC, il est indispensable de clarifier un peu la méthode employée par un ordinateur pour stocker des informations. Cette explication, un peu technique, facilitera la compréhension ultérieure de certaines fonctions du langage.

### *Le stockage des données*

Pour bien comprendre la manière dont un ordinateur manipule les informations, il est important de connaître à la fois le type de données qu'il utilise et leur mode de stockage en mémoire. Voyons rapidement les quelques principes de base :

1. L'unité d'information la plus élémentaire est le **BIT**, qui ne peut prendre que deux valeurs : 0 ou 1 : Il s'agit d'un nombre binaire. Pour gagner en vitesse et en efficacité, les Bits sont regroupés en "mots" de 8 bits, appelés **Octets**.

2. Un **Octet** peut prendre 256 ($2^8$) valeurs différentes, de 0 à 255 ; il constitue l'unité de stockage pour toutes les informations transitant par la mémoire, ainsi que la plus petite unité directement accessible à l'utilisateur. Certaines données, comme le code d'un caractère par exemple, peuvent se stocker sur un seul octet ; mais pour des valeurs supérieures à 255, deux octets ou plus sont nécessaires. Variables, programmes et constantes sont stockés sous la forme d'une suite d'octets dans la mémoire de l'ordinateur.

3. Chacun des huit bits constituant l'octet ne peut valoir que 0 ou 1, mais on attribue à chacun d'entre eux un "poids" différent, ce qui permet d'obtenir les 256 combinaisons citées plus haut. Ce principe est illustré par le schéma suivant :

| | OCTET | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Poids :* | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| *Rang :* | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Pour obtenir le poids d'un Bit, on élève 2 à la puissance symbolisée par le rang : 8 est égal à 2^3.

Pour obtenir la valeur totale d'un octet, on multiplie le poids de chacun des bits le constituant par sa valeur propre (0 ou 1), et on additionne le tout : Si, par exemple, les bits de "poids" 16, 8 et 1

sont à 1 et tous les autres à 0, l'octet aura la valeur 16+8+1=25. On voit que grâce à cette méthode de codage, un octet peut prendre toutes les valeurs entières comprises entre 0 (tous les bits à 0) et 255 (tous les bits à 1).

4. Pour savoir où est rangée chaque information et y accéder rapidement, l'ordinateur utilise des **Adresses**, numérotées entre 0 et 65535 : Chaque Octet est accessible par son adresse, qui permet de le définir par rapport à ses voisins. Par exemple, les octets constituant un programme sont rangés dans des adresses se suivant en mémoire. Deux commandes du Basic, PEEK et POKE, sont prévues pour lire et écrire directement dans les adresses mémoires (voir § 3.3.2).

**Résumé :**

La plus petite unité manipulée par un ordinateur est le **Bit**, valant 0 ou 1. Les Bits sont regroupés et codés sous forme d'**Octets**, dont la valeur peut varier de 0 à 255. Ces octets sont stockés dans des **Adresses** numérotées de 0 à 65535, chacune d'elles pouvant recevoir un octet. Un programme Basic, un calcul, une variable numérique et en règle générale toutes les données manipulées par un micro-ordinateur sont stockés dans des adresses sous la forme d'une suite d'octets.

## *Les données du MSX-BASIC*

Tous les types de données et d'opérateurs utilisés en MSX-BASIC ont été étudiés dans la première partie (§ 1.1 et 2). Cette section les récapitule en ajoutant les caractéristiques propres au standard MSX.

### 3.2.1. Les constantes numériques

Les constantes numériques sont les nombres utilisés par le Basic pour calculer. Elles peuvent appartenir à six différentes catégories :

**Entiers** : Un nombre entier est un nombre n'incluant pas de point décimal. Le MSX-BASIC peut traiter tous les nombres entiers positifs ou négatifs compris entre −32768 et 32767. Cette limite est imposée par le principe même du stockage des entiers en mémoire, sur deux octets (voir VARPTR § 3.4.1). 123, 6544, −55 sont des nombres entiers.

**Nombres décimaux** : Un nombre décimal est un nombre réél positif ou négatif pouvant prendre toute valeur entière ou décimale. En MSX-BASIC, la virgule est représentée par un point. 3.1416, 2.1, −9999.1 sont des nombres décimaux.

**Notation scientifique** : Un nombre trop grand ou trop petit pour être entier ou décimal peut être représenté en notation scientifique. Il se

compose alors de deux parties, un nombre entier ou décimal (la mantisse) suivi de la lettre E et d'un entier signé (l'exposant). La valeur du nombre est équivalente à celle de la mantisse que multiplie la puissance de 10 que représente l'exposant. En MSX-BASIC, la notation scientifique autorise le calcul sur des nombres compris entre 10 puissance −63 et 10 puissance +63.

***Exemple***: Le nombre entier 10205 s'écrit en notation scientifique 102.05 E2 ce qui équivaut à 102.05 × 10^2:

REMARQUE: Les nombres exprimés en décimal et en notation scientifique peuvent être spécifiés en simple et en double précision. La simple précision correspond à un stockage des constantes numériques sur six décimales, la double précision sur quatorze décimales. Par défaut, le MSX-BASIC travaille en double précision sur les nombres décimaux. On peut forcer l'un ou l'autre mode en utilisant des suffixes:

Pour un nombre décimal:

**!** spécifie la simple précision (222.3! par exemple).

# spécifie la double précision (75.12345# par exemple).

Pour la notation scientifique:

**E** correspond à la simple précision (22 E3).

**D** à la double précision (22.1 D-6).

**Notation hexadécimale**: Notre système de comptage habituel est décimal, probablement parce que nous avons dix doigts: Nous disposons de 10 signes, allant de 0 à 9. En informatique, on utilise souvent le système de notation Hexadécimale, plus adapté au calcul sur des octets (voir HEX$ § 3.4.1). On dispose dans ce cas de 15 signes, les dix premiers étant 0 à 9 et les cinq autres A, B, C, D, E et F. En décimal, A=10, B=11 jusqu'à F qui est égal à 15. Le Basic MSX reconnaît un nombre comme Hexadécimal s'il est précédé de "&H", et le convertit automatiquement en décimal. &H32, &H7F sont des nombres hexadécimaux.

**La notation Octale**: Comme pour les nombres hexadécimaux, exprimés en base 16, il est possible d'utiliser la base Octale pour traiter les valeur numériques. On dispose dans ce cas de 8 signes autorisés, allant de 0 à 7. Le préfixe "&O" sert à identifier ce type de notation. &O336 est un exemple de notation Octale.

**Les nombres binaires**: Lorsqu'on travaille avec des Octets, il est souvent plus simple de rentrer directement la valeur de chaque Bit plutôt que le résultat de la conversion en décimal. Le préfixe "&B" permet de traiter une suite de 0 et de 1 comme un nombre binaire:

&B10001011 est mémorisé directement par le Basic sous forme de l'octet correspondant, 139.

### 3.2.2. Les variables du MSX

Pour toutes les variables du MSX-BASIC, le nombre de caractères utilisé pour les nommer n'est pas limité, mais seuls les deux premiers sont pris en compte pour les différencier. Il est possible de mélanger chiffres et lettres dans un nom de variable, mais le premier caractère doit impérativement être une lettre.

Enfin, il faut faire attention à ne pas inclure dans un nom de variable tout ou partie d'une commande du Basic quelle qu'elle soit. On distingue quatre types de variables, associés à quatre suffixes différents :

Les variables numériques entières, suivies du signe **%**. A%, VALEUR% sont des noms de variables entières.

Les variables numériques en simple et en double précision, suivies comme les constantes des signes **!** et **#**. La double précision est sélectionnée par défaut si aucun suffixe n'est présent. COMPTE!, C# sont des noms autorisés.

Les variables chaînes de caractères, repérées par le signe **$**. On peut associer à une variable chaîne une autre variable ou une constante du même type dont la longueur maximum est 255 caractères. CHAINE$, A$ sont des noms de variables chaînes.

REMARQUES :

— Il est possible de convertir un nombre en le transférant dans une variable de la précision adéquate. Par exemple, l'égalité A%=2.356 convertira l'opérande en valeur entière pour le stocker dans A%. Cette règle est valable pour convertir le résultat d'une opération en double précision dans une variable en simple précision : A!=13/9.

— L'emploi de suffixes pour spécifier la précision n'est pas obligatoire, des instructions du Basic étant prévues pour effectuer systématiquement la conversion (voir DEFINT, DEFSTR, DEFDBL § 3.3.5).

— Les noms des tableaux de variables suivent les mêmes règles que les variables numériques et chaînes. En MSX-BASIC, le nombre maximum de dimensions possibles pour un tableau est 255. Le nombre d'éléments qui constituent le tableau n'est limité que par la taille mémoire disponible. L'instruction de déclaration DIM sert à fixer la dimension des tableaux (voir § 3.3.5).

— Des variables de types différents peuvent utiliser le même nom sans qu'il y ait confusion en mémoire : A!, A#, A(1,2), A%, A$ sont des variables différentes pour l'interpréteur du MSX-BASIC.

— Les variables entières sont stockées en mémoire sur deux octets, les variables simple précision sur quatre octets et les double précision sur huit octets. Dans un tableau numérique, chaque élément occupe l'espace correspondant au type de variable auquel le tableau appartient, et ce même si l'élément est vide.

— Les variables chaînes réservent en mémoire un nombre d'octets égal à la longueur de la chaîne plus trois octets (voir VARPTR § 3.4.1). Dans le cas d'un tableau de chaînes, la mémoire est allouée dynamiquement : Si un élément du tableau est vide, 3 octets seulement sont occupés en mémoire. La place mémoire varie ensuite avec la taille des chaînes que contient le tableau.

### 3.2.3. Les opérateurs arithmétiques

Déjà abordés au paragraphe 1.1.3, ceux-ci comprennent en dehors des quatre opérations de base, la négation, l'élévation à la puissance, la division entière et la division Modulo. En MSX-BASIC, l'ordre de priorité et la syntaxe se définissent comme suit :

| *Opérateur* | *Syntaxe* | *Fonction* |
|---|---|---|
| ^ | X^Y | Élève X à la puissance Y. B peut être un entier ou un nombre décimal. C'est la première opération exécutée par l'interpréteur dans une ligne de calcul. |
| – | -X | Calcul la négation du nombre X. |
| * | X*Y | Multiplication de X par Y. |
| / | X/Y | Division de X par Y. Le résultat est exprimé en double précision par défaut. |
| ¥ | X¥Y | Le symbole du Yen est utilisé pour la division entière. Les opérandes de la division X et Y sont convertis en entiers, la division est ensuite effectuée, puis le résultat est également converti en entier. X et Y doivent être compris entre –32768 et 32767. |
| MOD | X MOD Y | L'opération MOD renvoie le reste de la division entière de X par Y. Ce reste est lui-même un entier. |
| + | X+Y | Addition de X et Y. |
| – | X-Y | Soustrait Y à X. |

REMARQUES :

— Une division par 0, entière ou non, provoque une erreur et l'arrêt du programme.

— Si le résultat d'un calcul n'est pas compris entre 1 E—63 et 1 E63, une erreur de dépassement de capacité est provoquée. Il en va de même si les opérandes d'une division entière ou d'une opération modulo ne sont pas compris entre —32768 et 32767.

### 3.2.4. Les opérateurs relationnels

Six opérateurs relationnels peuvent être inclus dans une ligne de programme Basic :

| *Opérateur* | *Syntaxe* | *Fonction* |
|---|---|---|
| = | X=Y | X égal à Y |
| <> | X<>Y | X différent de Y |
| < | X < Y | X inférieur à Y |
| > | X > Y | X supérieur à Y |
| <= | X<=Y | X inférieur ou égal à Y |
| >= | X>=Y | X supérieur ou égal à Y |

Les opérateurs relationnels servent à comparer entre elles des valeurs numériques et des chaînes de caractères :

Dans le cas des nombres, les opérateurs relationnels tiennent compte des valeurs décimales et du signe pour comparer leurs valeurs.

***Exemples*** :

```
232.21  est supérieur à 232.11
-1222   est inférieur à   2
```

Dans le cas des chaînes, la comparaison s'effectue caractère par caractère en partant du premier élément à gauche de la chaîne. Le classement suit l'ordre croissant des codes ASCII, ce qui correspond à l'ordre alphabétique pour les lettres. Les majuscules ayant des codes inférieurs à ceux des minuscules, ces dernières seront traitées comme "supérieures" dans le cas d'une comparaison. Il est à noter que l'espace est considéré par un caractère de code ASCII 32 pour l'interpréteur du Basic. Le jeu de caractères du MSX est répertorié à l'appendice 1.

***Exemples*** :

```
"JEAN"  est inférieur à "jean"
"ABCD"  est supérieur à "ABBB"
```

Toute expression utilisant les opérateurs relationnels peut être assimilée à une opération renvoyant −1 si le résultat de la comparaison est "vrai" et 0 s'il est "faux":

```
IF A<>B THEN GOTO...  peut se traduire par SI (A DIFFERENT DE B)
                      EST VRAIE, ALORS ...
```

On aurait pu écrire directement:

```
IF (A<>B)=-1 THEN GOTO...
```

REMARQUE : Les calculs sont toujours effectués avant les comparaisons. Il est donc indispensable d'employer des parenthèses pour fixer des priorités entre comparaisons et expressions numériques.

### 3.2.5. Les opérateurs logiques

Les opérateurs logiques effectuent des opérations suivant les règles de la logique Booléenne. On les utilise pour comparer entre elles des valeurs numériques ou des expressions utilisant des opérateurs relationnels. Dans tous les cas, les opérations sont effectuées en **binaire**, c'est-à-dire bit à bit. On peut dresser une table des différents opérateurs logiques par ordre de priorité, avec la valeur du bit résultante pour toutes les combinaisons possibles :

**NOT:** Négation

| X | NOT X |
|---|---|
| 1 | 0 |
| 0 | 1 |

**AND:** ET logique

| X | Y | X AND Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

**OR:** OU logique

| X | Y | X OR Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**XOR:** OU exclusif

| X | Y | X XOR Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

**EQV:** Egalité logique

| X | Y | X EQV Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

**IMP:**

| X | Y | X IMP Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

**Combinaison d'opérateurs relationnels:**

On utilise fréquemment les opérations logiques pour affiner les tests avec les instructions conditionnelles IF et THEN du Basic. Dans ce cas, chaque opérateur à une signification correspondant à sa fonction très facile à mémoriser. Le plus simple est d'illustrer cette signification par des exemples pratiques:

**NOT**

```
IF NOT (A=200) THEN ... = IF A<>200 Then.
```

Si l'expression A=200 est fausse ALORS...

**AND**

```
IF A=100 AND B=20 THEN ...
```

Si les deux opérations A=100 et B=20 sont vraies en même temps...

**OR**

```
IF A=100 OR B=20 THEN ...
```

Si l'une des deux expressions A=100 ou B=20 (ou les deux simultanément) est vraie...

**XOR**

```
IF A=100 XOR B=20 THEN ...
```

Si l'une des deux expressions et une seule est vraie...

**EQV**

```
IF A=200 EQV B=20 THEN ...
```

Si les deux expressions sont vraies ou fausses en même temps...

**IMP**

```
IF A=200 IMP A/10=20 THEN ...
```

Si la première expression est vraie, la deuxième doit l'être également pour que le résultat soit vrai.

REMARQUES:

— Dans tous les cas, un 0 logique peut être assimilé à "faux" et un 1 logique à "vrai". On peut donc utiliser l'algèbre booléenne dans les expressions numériques puisque faux et vrai correspondent à 0 et −1 en Basic:

```
IF X AND Y THEN GOTO ...
```

Signifie SI (X ET Y) sont vraies en même temps ALORS...

On aurait pu écrire:

```
IF (X AND Y)=-1 THEN GOTO...
```

— Les opérations logiques viennent après les opérations arithmétiques et les comparaisons lors de l'évaluation d'une expression.

— Il est possible de combiner plusieurs opérateurs à l'intérieur d'un même test conditionnel:

```
IF (A=2 AND B=3) OR C=4 THEN ...
```

Dans ce cas, les expressions sont évaluées au fur et à mesure dans l'ordre des priorités.

**Les opérations binaires:**

Lorsqu'un opérateur logique est utilisé sur des nombres, il convertit les opérandes en binaire et effectue une comparaison Bit à Bit dont le

résultat est donné par la table du début de ce paragraphe. Par exemple, l'opération 8 AND 10 est équivalente à :

&B00001000 AND &B00001010 ce qui donne &B00001010 soit 10 en décimal.

Cette possibilité s'utilise surtout par les programmeurs travaillant beaucoup en langage-machine, mais reste peu utile dans les programmes n'utilisant que le Basic.

## 3.3. LES INSTRUCTIONS DU MSX-BASIC

Leur classement est effectué par ordre alphabétique, à l'intérieur de six grandes rubriques, correspondant aux principaux types de commandes du MSX-BASIC. Afin de faciliter la consultation de ce chapitre, voici la liste des instructions qui y sont présentées :

*Les commandes système*

AUTO
CONT
DELETE
LIST
LLIST
NEW
RENUM
RUN
TRON-TROFF

*Les commandes d'usage général*

CLEAR
DATA
END
INPUT
LET
LINE INPUT
LPRINT
LPRINT USING
OUT
POKE
PRINT
PRINT USING
READ
REM
RESTORE
STOP
SWAP
WAIT

*Boucles et branchements*

FOR/NEXT/STEP
GOSUB
GOTO
IF/THEN/ELSE
ON..GOSUB
ON..GOTO
RETURN

*Branchements automatiques*

INTERVAL ON/OFF/STOP
ON INTERVAL GOSUB
ON STOP GOSUB
STOP ON/OFF/STOP

*Les déclarations*

DEF FN
DEFDBL
DEFINT
DEFSGN
DEFSTR
DEFUSR
DIM
ERASE

*Le traitement des erreurs*

ON ERROR GOTO
RESUME
ERROR
ERL
ERR

**Rappel**: Les fonctions sont présentées au paragraphe 3.4, la gestion des périphériques, le graphique et le son aux chapitres 4, 5 et 6.

### 3.3.1. Les commandes système

Utilisées la plupart du temps en mode direct, les commandes systèmes regroupent les commandes plus particulièrement destinées à l'écriture et à la mise au point des programmes.

#### AUTO

*Syntaxe*: Auto [/N° de ligne début/,/Incrément/]

*Application*: Cette instruction d'aide à la programmation génère automatiquement des numéros de ligne à partir de /N° de ligne début/ avec un pas de /Incrément/. Ces deux paramètres doivent être des nombres entiers positifs.

REMARQUES:

— En cas d'omission des deux paramètres, AUTO 10,10 s'effectue par défaut.

— Si le numéro de ligne est suivi d'une virgule sans spécifier d'incrément, la dernière valeur utilisée pour l'incrément est prise par défaut.

— Si un numéro de ligne généré est identique à celui d'une ligne de programme déjà en mémoire, un astérisque (*) s'affiche à sa droite, vous indiquant que l'ancienne ligne sera effacée lors de la validation de la nouvelle. La frappe de RETURN permet de conserver la ligne non modifiée.

— On peut sortir du mode AUTO en tapant CONTROL/STOP (la ligne courante n'est pas sauvegardée).

*Voir aussi*: RENUM.

***Exemple***: En donnant à /N° de ligne début/ une valeur correspondant à la fin d'un programme déjà en mémoire, on peut compléter celui-ci tout en conservant la numérotation existante.

#### CONT

*Syntaxe*: CONT

STOP est dans le prog

*Application*: Redémarre un programme interrompu par l'instruction STOP, ou en pressant CONTROL/STOP.

REMARQUE: CONT ne fonctionne que si aucune modification n'a été apportée au programme pendant l'interruption.

*Voir aussi*: STOP.

**DELETE**

*Syntaxe*: DELETE [/N° ligne début/] [–/N° ligne fin/]

*Application*: Supprime les lignes comprises entre le numéro de début et le numéro de fin spécifié.

REMARQUES:

— Si on omet /N° de ligne fin/, DELETE supprime la ligne correspondant à /N° de début/.

— Si on omet /N° de début/, toutes les lignes jusqu'au /N° de ligne fin/ sont détruites.

— Si l'un des numéros de lignes n'existe pas, une erreur est provoquée.

— DELETE ne s'utilise qu'en mode direct, car elle entraîne l'interruption du programme.

*Voir aussi*: AUTO, RENUM.

***Exemple***: DELETE –1000 supprime toutes les lignes jusqu'à la ligne 1 000.

**LIST**

*Syntaxe*: LIST [/N° ligne début/] [–/N° ligne fin/]

*Application*: Liste à l'écran le programme en mémoire, en commençant par /N° ligne début/ jusqu'à /N° ligne fin/.

REMARQUES:

— Si tous les paramètres sont omis, le programme entier est listé sur l'écran.

— L'omission de /N° ligne fin/ et du tiret de séparation permet l'affichage d'une seule ligne de programme. Si le tiret est conservé, l'instruction liste le programme à partir de /N° ligne début/.

— Si c'est /N° ligne début/ qui est omis, tout le programme est listé jusqu'à /N° ligne fin/.

— Si les numéros de ligne spécifiés sont plus grands ou plus petits que les numéros existants, le listing se fera à partir des numéros les plus proches en ordre croissant ou décroissant.

— On peut suspendre temporairement le listing à l'aide de la touche STOP. Une deuxième pression reprend l'exécution au même point. L'action simultanée de CONTROL/STOP suspend définitivement l'exécution.

*Voir aussi*: LLIST

***Exemples***:

```
LIST 10-50 liste toutes les lignes comprises entre 10 et 50
LIST -100  liste du début du programme à la ligne 100
LIST 50-   liste de la ligne 50 à la fin du programme
```

**LLIST**

*Syntaxe*: LLIST [/N° ligne début/] [–/N° ligne fin/]

*Application*: Identique à LIST, cette instruction liste sur l'imprimante le programme présent en mémoire entre les deux limites spécifiées.

REMARQUE: Toutes les remarques concernant l'instruction LIST sont valables.

*Voir aussi*: LIST.

**NEW**

*Syntaxe*: NEW

*Application*: Supprime programme et variables de la mémoire de l'ordinateur.

*Voir aussi*: CLEAR, ERASE, DELETE.

**RENUM**

*Syntaxe*: RENUM [[/Nouveau N°/] [,[/Ancien N°/] [,/Incrément/]]]

*Application*: Renumérote les lignes d'un programme. La renumérotation commence à /Ancien N°/, qui prend alors la valeur de /Nouveau N°/. Les nouveaux numéros sont séparés par un pas correspondant à /Incrément/.

REMARQUES:

— Si /Ancien N°/ est omis, la numérotation débute à la première ligne du programme.

— Si c'est /nouveau N°/ qui est omis, la numérotation commence à 10.

— Si tous les paramètres sont omis, la renumérotation commence à la première ligne du programme, qui prend la valeur 10. L'incrément par défaut vaut également 10.

*Voir aussi*: AUTO, DELETE.

***Exemples***:

RENUM 20,,5 Renumérote les lignes à partir du début du programme. La nouvelle numérotation débute à 20 avec un pas de 5.

RENUM ,100 Le programme est renuméroté à partir de la ligne 100. La numérotation commence à 10 avec un pas de 10.

RENUM 100 Renumérote les lignes à partir du début du programme. La nouvelle numérotation commence à 10 avec un pas de 10.

RENUM ,,5 Fixe l'incrément à 5. Les autres paramètres sont pris par défaut.

**RUN**

*Syntaxe*: RUN [/N° ligne/]

*Application*: Démarre l'exécution du programme en mémoire au numéro de ligne spécifié. Si ce paramètre est omis, le programme débute au plus petit numéro existant.

REMARQUE: RUN efface toutes les variables de la mémoire.

*Voir aussi*: GOTO, CLEAR.

**TRON/TROFF**

*Syntaxe*: TRON
TROFF

*Application*: TRON liste à l'écran entre crochets tous les numéros des lignes exécutées en cours de programme. Cette instruction fournit une aide précieuse à la mise au point, la détection des erreurs s'effectuant simplement en suivant pas à pas le cheminement du programme. TROFF permet le retour au mode normal.

REMARQUES:

— TRON peut aussi bien être inséré dans un programme qu'utilisé en mode direct.

— Les numéros de ligne s'affichant à la position courante du curseur qui varie souvent au cours d'un programme, l'écran devient vite illisible en mode TRON. Pour résoudre ce problème, il suffit d'appuyer de temps en temps sur la touche STOP pour interrompre l'exécution et la relancer.

***Exemple***: Suivi d'une boucle FOR/NEXT en mode TRON:

```
10 TRON
20 CLS
30 FOR I= 1 TO 50
40 PRINT " LIGNE :";
50 NEXT I
60 TROFF
```

### 3.3.2. Les commandes d'usage général

Cette rubrique réunit toutes les instructions du Basic d'usage général.

**CLEAR**

*Syntaxe*: CLEAR [/Espace chaîne/[,/Haut de la mémoire /]]

*Application*: Réinitialise toutes les variables et clos tous les fichiers. Les variables numériques prennent toutes la valeur 0, et les variables chaînes sont vidées.

REMARQUE: Deux options sont disponibles dans la syntaxe de l'instruction:

— /Espace chaîne/ fixe la taille mémoire allouée aux chaînes de caractères. A la mise sous tension, 200 octets seulement sont disponibles pour créer des chaînes de caractères, ce qui correspond à une seule constante chaîne de longueur 200 ! Il est donc indispensable d'initialiser CLEAR avec une valeur plus élevée pour pouvoir travailler correctement avec des chaînes.

— /Haut de la mémoire/ définit la taille de la mémoire utile, en fixant la limite supérieure de la zone accessible par le Basic. Pour comprendre le principe, il faut savoir que le Basic permet d'adresser 65536 adresses mémoires. La valeur la plus élevée correspond à la dernière adresse accessible par le langage. Pour écrire des programmes en langage-machine, on utilise des portions de la mémoire dans lesquelles on "range" des suites d'octets. Il est intéressant de réserver une section de mémoire interdite au Basic, dans laquelle ces programmes seront protégés d'une fausse manœuvre. /Haut de la mémoire/ réserve un espace en abaissant la valeur de la dernière adresse autorisée au Basic.

*Voir aussi*: RUN.

***Exemples***:

| | |
|---|---|
| `CLEAR 2000` | Efface toutes les variables et réserve deux kilo-octets pour les chaînes de caractères. |
| `CLEAR,60000` | Efface toutes les variables et abaisse de 5 535 octets le haut de la mémoire. |

**DATA**

*Syntaxe*: DATA /Constante/[,/Constante/...]

*Application*: DATA permet de stocker une série de constantes numériques ou chaînes, qui pourront être relues et affectées à des variables par l'instruction READ. /Constante/ peut être un nombre ou une chaîne

de caractères, mais pas une expression numérique. Dans le cas des chaînes, les guillemets doivent en général être omis, sauf si elles contiennent des virgules ou des espaces en début ou en fin.

REMARQUES :

— Le nombre de constantes, séparées par des virgules, contenu par une ligne de DATA n'est limité que par la taille maximum de la ligne de programme en Basic (255 caractères).

— Une ligne contenant une instruction DATA est ignorée par le MSX-BASIC lors de l'exécution du programme, et peut donc être placée à n'importe quel endroit de celui-ci.

— Si plusieurs lignes contiennent des DATA, la lecture avec READ se fera dans l'ordre croissant des numéros de ligne (voir aussi cette instruction pour exemples pratiques).

*Voir aussi*: READ, RESTORE.

***Exemple*** :

```
10 DATA 35,BONJOUR,AU REVOIR,22.54
20 READ A!,NOM$,X$,B: REM LECTURE DES DATA PAR READ
30 PRINT A!,NOM$,X$,B: REM AFFICHAGE  DES  VARIABLES
```

## END

*Syntaxe*: END

*Application*: Indique la fin d'un programme. Le message "OK" est affiché à la position du curseur. L'omission de END ne provoque pas d'erreur.

REMARQUE : A la différence de STOP, une exécution interrompue par END ne peut être reprise à l'aide de CONT.

*Voir aussi*: STOP.

## INPUT

*Syntaxe*: INPUT ["/Message/";]/Variable/[,/Variable/...]

*Application*: Lorsqu'une instruction INPUT est rencontrée dans le programme, celui-ci s'arrête momentanément et affiche un point d'interrogation à la position du curseur pour indiquer qu'il attend des données. Si un message est inclus dans l'instruction, il est affiché avant le point d'interrogation. /Variable/ est un nom de variable quelconque, numérique, chaine, ou tableau à laquelle sera affectée la valeur frappée au clavier.

REMARQUES :

— L'entrée est validée par RETURN. Si aucune donnée n'est saisie, les variables numériques valent O et les chaînes sont vides (" ").

— La liste qui suit INPUT peut contenir un nombre quelconque de variables de types différents (chaînes, numériques, tableaux), mais toutes doivent être séparées par des virgules.

— Le type et l'ordre des données saisies doivent correspondre au type et à l'ordre des variables de la liste suivant l'instruction INPUT, faute de quoi un message d'erreur est affiché et l'entrée annulée.

— Lors de l'entrée des données, une virgule doit séparer chacune d'entre elles de la suivante.

— Si le nombre de données saisies est supérieur au nombre de variables de la liste, un message d'erreur est affiché et les données en excédent sont ignorées.

— Si ce nombre est inférieur, un double point d'interrogation informe l'utilisateur qu'il doit continuer à entrer des données.

*Voir aussi* : LINE INPUT, INPUT$, INKEY$.

***Exemple*** : De très nombreux exemples sont donnés dans la première partie consacrée à l'initiation au Basic, § 2.3 :

## LET

*Syntaxe* : [LET]/Variable/=/Constante/

*Application* : Assigne à /Variable/ la valeur de /Constante/. Ces deux paramètres doivent évidemment être du même type, numérique ou chaîne.

REMARQUE : LET peut être omis.

*Voir aussi* : SWAP.

***Exemple*** :

```
LET VALEUR=50 est équivalent à VALEUR=50.
```

## LINE INPUT

*Syntaxe* : LINE INPUT ["/Message/";]/Variable chaîne/

*Application* : LINE INPUT permet d'assigner à une variable une chaîne de caractères, quelque soit son contenu. Tous les séparateurs habituellement reconnus par INPUT sont acceptés, la saisie ne peut se terminer qu'avec [RETURN] ou CONTROL/STOP.

REMARQUES :

— Le message est optionnel, et n'est pas suivi d'un point d'interrogation.

— Jusqu'à 254 caractères peuvent être saisis par cette instruction.

*Voir aussi* : INPUT, INPUT$, INKEY$.

***Exemple*** : Saisie d'une phrase contenant des séparateurs :

```
10 LINE INPUT "ENTREZ NOM, AGE ET ADRESSE :";N$
```

essayer sans line

Affiche sur l'écran :

```
ENTREZ NOM, AGE ET ADRESSE : _
```

On peut répondre :

```
DUVAL, 30 ANS, 5 RUE DES ACACIAS, 75012 PARIS [RETURN]
```

## LPRINT

*Syntaxe* : LPRINT [/Liste de données/]

*Application* : Cette instruction d'affichage est totalement équivalente à PRINT pour l'imprimante. Les séparateurs "," ou ";" fonctionnent de manière identique.

REMARQUES :

— Les fonctions TAB et SPC fonctionnent également avec LPRINT.

— Si l'imprimante n'est pas connectée ou en mode "ready", le déroulement du programme est bloqué par LPRINT.

*Voir aussi* : PRINT, PRINT USING, LPRINT USING, TAB, SPC.

***Exemple*** : `LPRINT "BONJOUR";" MONSIEUR"` écrit **BONJOUR MONSIEUR** sur l'imprimante :

## LPRINT USING

*Syntaxe* : LPRINT USING "/Chaîne de caractère/";/Liste de données/

*Application* : LPRINT USING sert à envoyer des données formatées à l'imprimante : Les règles d'utilisation sont identiques à celles de PRINT USING.

*Voir aussi* : PRINT, PRINT USING, LPRINT, TAB, SPC.

## OUT

*Syntaxe* : OUT /N° de port/,/Constante entière/

*Application* : Toute action à partir du clavier vers l'écran, ou à partir d'un quelconque périphérique vers l'unité centrale, passe par les ports d'entrée-sortie du Z 80. OUT est une instruction permettant le contrôle direct du système d'entrée-sortie de l'ordinateur. On l'utilise pour envoyer /Constante/ sur le port spécifié du Z 80. /N° de port/ et /Constante/ doivent être des entiers compris entre 0 et 255.

REMARQUE : OUT s'utilise principalement dans le cadres de programmes faisant appel à des fonctions spécifiques à une machine donnée. Son emploi nécessite une connaissance précise de l'architecture interne de l'ordinateur et ne concerne pas les utilisateurs du Basic.

*Voir aussi* : INP.

## POKE

*Syntaxe* : POKE /Adresse mémoire/,/Octet/

*Application* : Cette instruction sert à effectuer des accès directs à la mémoire, c'est-à-dire lire et écrire directement dans des adresses. Le contenu de /Adresse mémoire/ est remplacé par l'Octet spécifié. /Adresse mémoire/ doit être un entier compris entre 0 et 65535. /Octet/ doit lui être compris entre 0 et 255.

REMARQUE : Une utilisation erronée de POKE peut bloquer le fonctionnement de l'ordinateur.

*Voir aussi* : PEEK.

***Exemple*** : Pour programmer en langage-machine, on a très souvent besoin de stocker une adresse en mémoire ; deux octets sont nécessaires dans ce cas, dans la mesure où la valeur d'une adresse peut être comprise entre 0 et 65535 donc supérieure à 255. Un des deux octets, dit "de poids fort" contiendra le quotient entier de la division [Adresse à coder]/256, et l'autre octet dit "de poids faible" le reste de cette opération. Ces deux octets sont ensuite placés dans deux adresses consécutives, celui de poids faible en premier.

Supposons par exemple que l'on veuille stocker le nombre 3560 aux adresses 1000 et 1001 : il faudra placer la partie entière de la division 3560/256, donc 13 en 1001, et le reste (232) en 1000. Pour ce faire, on écrira :

```
POKE 1000,232
POKE 1001,13
```

Pour relire le contenu de l'adresse, on utilise PEEK qui effectue l'opération inverse de POKE.

## PRINT

*Syntaxe* : PRINT [/Liste de données/]

*Application* : Inscrit /Liste de données/ à l'écran, à la position courante du curseur. Les données à imprimer peuvent être des constantes ou des expressions numériques, des chaînes de caractères ou encore n'importe quel type de variables.

Ces données peuvent être constituées d'un seul ou plusieurs éléments séparés par des ";" ou des ",". Un ";" affiche les éléments les uns à la suite des autres, alors que "," place l'élément qui suit à la position de tabulation suivante (une position toutes les quatorze colonnes). Si un ";" termine la série de données, le curseur demeure à la fin de celles-ci, alors que dans le cas contraire un passage à la ligne suivante est effectué.

REMARQUES :

— PRINT peut être remplacé par un point d'interrogation dans un programme (?).

— PRINT sans argument permet de sauter une ligne.

— Le point-virgule est facultatif entre des variables ou des chaînes de caractères :

```
PRINT "valeur :";A est équivalent à  PRINT "valeur :"A

PRINT NOM$;" ";PRENOM$ équivaut à PRINT NOM$" "PRENOM$
```

— Un espace est toujours ajouté à droite des valeurs numériques. Si le nombre est négatif, un signe "–" le précède, s'il est positif, il est remplacé par un espace.

— Toutes les commandes de l'éditeur du MSX-BASIC peuvent être utilisées par programme, en affichant avec PRINT le caractère correspondant à la fonction (voir CHR$, les codes de contrôles).

*Voir aussi* : PRINT USING, LOCATE, TAB, SPC.

***Exemples*** :

```
10 REM AFFICHAGE SIMULTANE CHAINE+NOMBRE
20 CLS
20 FOR I=1 TO 10
30 PRINT "COMPTEUR I=";I
40 NEXT I
```

## PRINT USING

*Syntaxe* : PRINT USING "/Constante chaîne/";/Liste de données/

*Application* : L'option USING de l'instruction PRINT permet l'affichage de nombres et de chaînes de caractères selon un format défini dans /Constante chaîne/ à l'aide de caractères spéciaux. Cette commande très puissante autorise un contrôle complet de la présentation écran et

des paramètres d'affichage. On l'utilise principalement pour aligner des tableaux de nombres sur le point décimal ou pour spécifier la portion d'une chaîne qui doit être imprimée.

**Affichage d'une chaîne de caractères :**

— Si "/Constante chaîne/"="!", seul le premier caractère de la chaîne donnée en argument est affiché.

```
10 NOM$="DUVAL"
20 PRINT "INITIALE : ";
30 PRINT USING "!";NOM$
```

Donnera : **INITIALE : D**

Comme pour PRINT, plusieurs variables peuvent être données en argument. Dans ce cas, l'instruction de format s'applique à toutes les chaines à afficher :

```
10 NOM$="DUVAL"
20 PRENOM$="JEAN"
30 PRINT "INITIALES : ";
40 PRINT USING "!";NOM$;PRENOM$
```

Donnera : **INITIALES : DJ**

Le point-virgule entre les chaînes est également facultatif. On aurait pu écrire pour la ligne 40 :

```
40 PRINT USING "!";NOM$"."PRENOM$
```

Pour obtenir : **INITIALES : D.J**

— Si "/Constante chaîne/"="& &", seule une portion de la chaîne à imprimer correspondant à la longueur de cette constante est affichée. Le nombre d'espaces séparant les deux Ampersands permet de régler la longueur de cette portion, qui ne peut être inférieure à deux (aucun espace entre les Ampersands) :*

```
10 NOM$="DUVAL":PRENOM$="JEAN"
20 PRINT USING "& &";NOM$;" ";PRENOM$
```

Donnera : **DUV JEA**

Si la chaîne à afficher est plus petite que la longueur spécifiée par l'instruction, des espaces sont ajoutés à sa droite :

```
10 NOM$="JEAN DUVAL"
20 PRINT USING "&&";NOM$
30 PRINT USING "&  &";NOM$
40 PRINT USING "&        &";NOM$
50 PRINT USING "&              &";NOM$
```

* Sur certaines machines MSX, le caractère "&" doit être remplacé par "\"

Donnera:

```
JE
JEAN
JEAN DU
JEAN DUVAL
```

— Si "/Constante chaîne/" contient une chaîne incluant le caractère "@", la chaine donnée en argument est *insérée* à la place de ce caractère dans la constante, et c'est cette dernière qui est affichée:

```
10 NOM$="JEAN"
20 PRINT USING "VOUS VOUS APPELEZ @ DUVAL.";NOM$
```

Donnera sur l'écran: **VOUS VOUS APPELEZ JEAN DUVAL.**

**Affichage des valeurs numériques:**

Pas moins de neuf caractères spéciaux peuvent être inclus dans une /Constante chaîne/ pour imposer le format des nombres à l'affichage. Dans les exemples qui suivent, des constantes numériques sont utilisées dans les exemples par souci de clarté, il est bien entendu possible d'employer des variables:

— Le caractère "#" sert à visualiser le nombre de caractères à afficher. Il suffit de décrire dans /Constante chaîne/ le format désiré, en incluant si nécessaire le point décimal. L'instruction justifie le nombre à afficher à droite et effectue l'arrondi automatiquement:

```
PRINT USING "###";10.37          donne   10
PRINT USING "###.#";10.37        donne   10.4
```

Si des décimales sont demandées et que le nombre n'en comprend pas, des 0 sont affichés à sa droite:

```
PRINT USING "###.##";20,10.7     donne   20.00      10.70
```

Si le nombre à visualiser est plus grand que le champ spécifié, il est affiché en entier avec un "%" à sa gauche pour indiquer que le format est dépassé:

```
PRINT USING "###";1000           donne   %1000
PRINT USING ".##";2              donne   %2.00
```

On peut entrer au maximum 24 positions avec "#" dans un PRINT USING. Au dessus, une erreur est provoquée.

— Le signe "+" à droite ou à gauche de la constante chaîne impose l'affichage du signe du nombre au même emplacement que ce caractère:

```
PRINT USING "+##.##";2.3,-7        donne  +2.30      -7.0
PRINT USING "##.##+";2.3,-7        donne  2.30+      7.00-
```

— Un signe "–" à droite de la constante chaîne provoquera l'affichage d'un signe moins à droite des nombres négatifs:

```
PRINT USING "##.##-";2.3,-7        donne   2.30      7.00-
```

— Le signe "**" à gauche de la constante chaîne rajoute deux positions d'affichage et impose le remplissage des positions non utilisées avec des astérisques:

```
PRINT USING "**##.##";1.5                 donne   ***1.50
```

— Une virgule à gauche du point décimal dans /Constante chaîne/ provoque l'affichage d'une virgule tous les blocs de trois chiffres (dans la partie entière uniquement):

```
PRINT USING "####,.##";1550               donne   1,550.00
```

Si la virgule est entrée à la droite de /Constante chaîne/, elle s'affiche à l'écran à droite du nombre:

```
PRINT USING "####.##,";1550;122.3  donne  1550.00, 122.3
```

— On peut imposer l'affichage en notation scientifique en faisant suivre la chaîne de format de la constante "^^^^". Le nombre est alors converti en respectant les positions spécifiées par "#" pour la mantisse:

```
PRINT USING "##.##^^^^";222.7             donne   2.23E+02
```

REMARQUES:

— On peut combiner entre eux les divers formats, exceptée la notation scientifique et la virgule. Dans l'exemple qui suit, la constante chaîne inclus un formatage pour des caractères suivi d'un autre pour des nombres:

```
10 CLS
20 M$="NOMBRE:"
30 INPUT "ENTREZ UN NOMBRE ";A:REM           SAISIE D'UN NOMBRE
40 PRINT USING "&       &+###.##";M$;A:REM   AFFICHAGE COMMENTE
50 GOTO 30
```

remplacer & par \

Il est indispensable de choisir pour la liste de variables un ordre correspondant aux instructions de format, faute de quoi une erreur est provoquée.

— /Constante chaîne/ peut être remplacée par une variable, à condition de respecter la syntaxe de PRINT USING:

```
10 NOM$="DUVAL"
20 PRENOM$="JEAN"
30 CHAINE$="!"
30 PRINT "INITIALES : ";
40 PRINT USING CHAINE$;NOM$;".";PRENOM$
```

Donnera **INITIALES :D.J**

*Voir aussi*: PRINT, LOCATE, LPRINT USING.

***Exemple***: Mise en forme d'un tableau de nombres:

```
10 CLS
20 PRINT "SAISIE D'UN TABLEAU DE 10 NOMBRES: "
30 PRINT
40 FOR I=0 TO 9
50 INPUT "ENTREZ UN NOMBRE ";A(I)
60 NEXT I
70 PRINT
80 PRINT "AFFICHAGE DU TABLEAU: "
90 PRINT
100 M$="NOMBRE "
110 FOR I=0 TO 9
120 PRINT USING "&       &##";M$;I+1;
130 PRINT USING "&&+#####.##";" ";A(I)
140 NEXT I
```

**Explications :**

— La boucle des lignes 40-60 saisit dans le tableau numérique A( ) une série de 10 nombres.

— Celle des lignes 110-140 sert à afficher le contenu du tableau. La ligne 120 visualise le numéro de chaque élément et la ligne 130 le nombre correspondant. Dans les deux cas, PRINT USING affiche simultanément une chaîne et un nombre.

## READ

*Syntaxe*: READ /Variable/[,/Variable/...]

*Application*: READ affecte à une liste de variables les constantes contenues dans les lignes de DATA. Les variables peuvent être numériques ou chaînes, mais leur type doit correspondre à celui des constantes lues dans les DATA.

REMARQUES:

— Lorsque tous les DATA ont été lus dans un programme, l'emploi de READ provoque un message d'erreur (voir RESTORE).

— Les lignes de DATA sont lues dans l'ordre des numéros de ligne, en commençant au début du programme (après exécution de READ, un pointeur mémorise la position du dernier DATA lu).

*Voir aussi*: DATA, RESTORE.

***Exemple***: Dans cet exemple, la ligne 50 effectue une lecture des DATA dans deux tableaux A$( ) et A( ). L'ordre de lecture est imposé par la ligne 80: Une chaîne suivie d'un nombre:

```
10 CLS
20 PRINT "SAISIE ET AFFICHAGE DE 5 NOMS ET AGES: "
30 PRINT
40 FOR I=0 TO 9
50 READ A$(I):READ A(I)
60 PRINT "NOM: ";A$(I);"  AGE:";A(I);" ANS"
70 NEXT I
80 DATA JEAN,20,LOUIS,32,LEON,55,ROGER,30,ALAIN,21
```

**REM**

*Syntaxe*: REM/Ligne de commentaire/

*Application*: Insertion de commentaires non exécutables dans un programme. Chaque fois qu'une ligne de programme contient une instruction REM, la partie de la ligne à droite de la commande est ignorée par l'interpréteur.

REMARQUES:

— Les remarques sont affichées sans transformation par LIST. On se sert de cette possibilité pour clarifier les listings des programmes.

— Le caractère "'" peut remplacer REM et :REM. La ligne suivante:

```
10 PRINT "BONJOUR":REM AFFICHE A L'ECRAN
```

peut être remplacée par:

```
10 PRINT "BONJOUR" '  AFFICHE A L'ECRAN
```

**RESTORE**

*Syntaxe*: RESTORE [/N° de ligne/]

*Application*: Lorsque tous les DATA ont été lus dans un programme, l'exécution d'un READ provoque une erreur. La commande RESTORE sert à réinitialiser les DATA pour une relecture éventuelle.

Si /N° de ligne/ est omis, RESTORE replace le pointeur des DATA sur la première ligne de programme les contenant. La lecture avec READ se fait ensuite à partir du pointeur. Si une valeur est spécifiée pour /N° de

ligne/, le pointeur est placé sur la ligne de DATA ayant ce numéro, ou le suivant immédiatement.

REMARQUE : /N° de ligne/ peut être une constante ou une variable, mais doit être une valeur entière.

*Voir aussi* : READ, DATA.

***Exemple*** :

```
10 CLS
20 READ A$:PRINT A$
30 FOR I=0 TO 4
40 READ A
50 PRINT A;" ";
60 NEXT I
70 PRINT
80 RESTORE 110
90 GOTO 30
100 DATA AFFICHAGE DE NOMBRES
110 DATA 10,20,30,40,50
```

Ce petit programme relit en boucle les DATA de la ligne 110. La ligne 100 n'est lue qu'une fois, au début du programme.

## STOP

*Syntaxe* : STOP

*Application* : Permet l'interruption du déroulement du programme. Après exécution d'un STOP, un message indiquant le numéro de la dernière ligne exécutée est imprimé à la position du curseur :

BREAK IN /N° de ligne/

REMARQUES :

— Il est possible de reprendre l'exécution avec CONT si aucune modification n'est apportée au programme durant l'interruption.

— L'instruction STOP est équivalente à une action simultanée des touches CONTROL et STOP du clavier.

— Toutes les variables sont conservées par un STOP. On peut donc employer cette commande lors de la mise au point pour suivre le déroulement d'un programme, en insérant des points d'arrêt en différents endroits du Listing.

*Voir aussi* : CONT, END.

## SWAP

*Syntaxe* : SWAP /Variable/,/Variable/

*Application*: Échange le contenu de deux variables. Les deux variables peuvent appartenir à n'importe quelle catégorie, mais doivent impérativement être de même type.

REMARQUE: SWAP est très utile pour trier des nombres.

*Voir aussi*: LET.

***Exemples***:

```
100 ' TRI RIPPLE-SORT D'UNE LISTE DE 5 NOMBRES
110 DATA 5,12,9,8,4
120 FOR I=0 TO 4
130 READ A(I) '        LECTURE D'UN TABLEAU A()
140 NEXT I
150 FOR J=0 TO 4
160 FOR I=J TO 4
170 IF A(J)>A(I) THEN SWAP A(I),A(J)
180 NEXT I
190 PRINT A(J), '        AFFICHE LA LISTE TRIEE
200 NEXT J
```

**Explications:**

— 110,140: Une liste quelconque de nombres est lue dans un DATA et placée dans le tableau A( ).

— 150,200: Chaque élément du tableau est successivement comparé à tous les autres, et le plus petit des deux est placé en début de liste.

Cet algorithme de tri, dit RIPPLE-SORT est très simple à mettre en œuvre, mais n'est pas très performant pour des tris portant sur des listes importantes. La méthode de Shell-Metzner est l'une des plus rapides existantes, pour les tableaux de taille moyenne (jusqu'à 500 éléments environ):

```
60000 ' TRI SHELL-METZNER
60005 ' A()=Tableau à trier, T=Taille de A()
60007 '
60010 P=T
60020 P=INT(P/2):IF P<1 THEN RETURN
60030 N=1:K=T-P
60040 M=N
60050 H=M+P
60060 IF A(M)<=A(H) THEN 60090
60070 SWAP A(M),A(H)
60080 M=M-P:IF M<1 THEN 60090 ELSE 60050
60090 N=N+1:IF N>K THEN 60020 ELSE 60040
```

Ce programme trie un tableau nommé A( ), mais il peut aussi bien trier des tableaux de chaînes (classement alphabétique) ou d'entiers. En retour de sous-programme, le tableau A( ) est trié par ordre croissant, le

premier élément étant le plus petit et le dernier le plus grand. La taille T du tableau à trier doit être fournie à la routine par le programme principal.

Nous n'entrerons pas dans les détails du fonctionnement de cet algorithme, dont le principe est assez complexe. Disons simplement qu'au début, le tableau est divisé en deux parties, chaque élément de la partie basse étant comparé à l'élément correspondant de la partie haute. Si ce dernier est plus petit, ils sont inversés. Lorsque la comparaison entre les deux parties est terminée, on divise chacune d'entre elle par deux et on reprend la comparaison. Lorsque le pas (P) séparant deux éléments est égal à 1, le tableau est totalement trié.

## WAIT

*Syntaxe*: WAIT /N° de port/,/Octet/[,/Octet2/]

*Application*: Interrompt le déroulement du programme jusqu'à détection d'un Octet sur le numéro de port du Z 80 spécifié (voir OUT). La valeur de /Octet/ est alors comparée Bit à Bit par AND avec l'octet lu sur le port. Si le résultat est différent de 0, le déroulement du programme est repris, sinon la lecture est effectuée en boucle. Si l'argument optionnel /Octet2/ est inclus dans l'expression, un XOR avec /Octet2/ vient modifier l'octet lu avant la comparaison.

REMARQUE: /N° de port/, /Octet/ et /Octet2/ sont des entiers compris entre 0 et 255.

*Voir aussi*: OUT, INP.

### 3.3.3. Boucles et branchements

On a déjà vu dans la première partie comment écrire des boucles avec FOR/NEXT, et contrôler le déroulement d'un programme avec GOTO et GOSUB (voir § 2.6 et 2.7). Cette section reprend ces instructions en étendant leur domaine d'application.

## FOR/TO/NEXT/STEP

*Syntaxe*:

```
FOR N=/Valeur initiale/ TO /Valeur finale/ [STEP S]
.
.
/Traitement/
.
.
NEXT [N]
```

*Applications*: Ces commandes permettent la création d'une boucle, à l'intérieur de laquelle une suite d'instructions peut être exécutée un certain nombre de fois : Au démarrage, la variable **N** est égale à /Valeur initiale/. La section du programme qui suit l'instruction FOR est exécutée jusqu'à rencontre de l'instruction NEXT. La valeur de **N** est alors incrémentée de **S**, et l'exécution est reprise à la ligne suivant FOR. Lorsque **N** est égal ou supérieur à /Valeur finale/, le programme passe à la ligne suivant NEXT. Le nombre de répétitions est donc égal à (1+/Valeur finale/–/Valeur initiale/)/S.

REMARQUES :

— N et S peuvent être des variables entières, simple ou double précision, mais seul le choix de valeurs entières optimise la vitesse d'exécution.

— /Valeur initiale/ et /Valeur finale/ peuvent être des constantes ou des expressions numériques.

— Si l'instruction STEP est omise, S est égal à 1 par défaut.

— La variable N peut être omise dans NEXT. Dans ce cas, chaque NEXT est associé au premier FOR qui le précède.

— Après exécution de la boucle, N est égal à /Valeur finale/+S.

— /Valeur initiale/ peut être supérieure à /Valeur finale/ en spécifiant un pas négatif pour STEP.

— Un FOR doit toujours être suivi de NEXT, sinon une erreur est provoquée.

— On peut imbriquer des boucles entre elles (voir exemples).

***Exemples*** :

```
100 REM BOUCLE SIMPLE
110 FOR I=1 TO 100
120 PRINT I;
130 NEXT I
```

L'emploi de variables entières diminue notablement le temps d'exécution :

```
100 REM BOUCLE OPTIMISEE
110 FOR I%=1 TO 10  STEP 0.1
120 PRINT I%;
130 NEXT I%
```

pas de ,

```
100 REM INCREMENT NEGATIF
110 FOR I= 100 TO 1 STEP -1
120 PRINT I;
130 NEXT I
```

```
100 REM BOUCLES IMBRIQUEES
110 FOR I%=66 TO 90 STEP 2
120 FOR J%=1 TO 10
130 PRINT CHR$(I);J
140 NEXT J%
150 NEXT I%
```

Le nombre de boucles imbriquées n'est limité que par la taille mémoire disponible sur l'ordinateur.

REMARQUE : Les lignes 140 et 150 ne peuvent être interverties, sous peine de provoquer un message d'erreur. MSX-BASIC autorise l'écriture sous une forme condensée pour ces deux lignes :

```
140 NEXT J%,I%
```

## GOSUB/RETURN

*Syntaxe* : GOSUB /N° de ligne/

*Application* : Effectue un branchement vers le sous-programme débutant à /N° de ligne/. Le sous-programme doit se terminer par RETURN, pour permettre un retour au programme principal. L'exécution est alors reprise à la ligne suivant le dernier GOSUB rencontré. Un sous-programme peut être placé n'importe où, mais la rencontre d'une instruction RETURN sans que le branchement ait été préalablement effectué par GOSUB provoque une erreur

REMARQUES :

— Les sous-programmes peuvent en appeler d'autres à leur tour, et leur nombre n'est limité que par la taille de la pile de l'ordinateur (La pile est une zone mémoire servant à stocker les numéros de ligne où l'exécution reprend après RETURN).

— /N° de ligne/ peut être une expression numérique contenant des variables, pour un branchement calculé.

— La multiplication des GOSUB est un moyen simple de structurer les programmes. Elle permet des modifications aisées de l'organisation interne, en changeant des blocs de lignes sans avoir à redéfinir tout le programme, et le réemploi ultérieur de routines fréquemment utilisées sans les réécrire.

*Voir aussi* : ON...GOSUB, IF/THEN/ELSE.

***Exemple*** : Appel imbriqué de deux sous-programmes par un petit programme source (lignes 100-130) :

```
100 REM DEMO GOSUB
110 GOSUB 1000
120 PRINT "TERMINE"
130 END
1000 REM SOUS-PROGRAMME BOUCLE
1010 FOR I= 1 TO 100
1020 GOSUB 2000
1030 NEXT I
1040 RETURN
2000 REM SOUS-PROGRAMME AFFICHE
2010 PRINT I;
2020 RETURN
```

## GOTO

*Syntaxe*: GOTO /N° de ligne/

*Application*: GOTO effectue un branchement inconditionnel vers la ligne spécifiée. Contrairement à GOSUB, l'emploi de GOTO est en contradiction avec la clarté des listings. Il est conseillé de n'utiliser cette instruction que pour créer des boucles ou lorsque l'emploi d'un sous-programme est impossible.

REMARQUES :

— GOTO est très utile en mode direct pour essayer des parties d'un programme lors de sa mise au point.

— Comme pour GOSUB, /N° de ligne/ peut être une expression numérique contenant des variables.

*Voir aussi*: ON...GOTO, IF/THEN/ELSE.

***Exemple*** :

```
10 X$=INKEY$
20 IF X$=" " THEN GOTO 1000
30 PRINT X$
40 GOTO 10
1000 PRINT "TERMINE"
```

Ce petit programme saisit et affiche les caractères frappés au clavier, jusqu'à l'entrée d'un espace (voir INKEY$, § 3.4.1).

## IF/THEN/ELSE

*Syntaxes*:

```
                     [AND
IF /Condition/          /Condition/] THEN /Instructions/
                     [OR

IF /Condition/[(..)] THEN /Instructions/ ELSE /Instructions/
```

*Applications*: On appelle ces instructions les branchements conditionnels. Elles permettent la prise d'une décision en fonction du résultat d'un test ou d'un calcul.

— Dans sa première syntaxe, la ligne de programme exécutera la série d'instructions située après THEN si la condition résultant de l'expression située entre IF et THEN est VRAIE (=−1), sinon l'exécution du programme continue à la ligne suivante.
— Dans la deuxième syntaxe, les instructions situées après THEN seront exécutées si /condition/ est vraie (=−1). Si /condition/ est fausse (=0), le programme exécutera la série d'instructions suivant ELSE.

REMARQUES:
— Tous les opérateurs relationnels et logiques du MSX-BASIC peuvent être employés pour affiner la condition située entre IF et THEN. Les règles d'utilisation sont récapitulées au paragraphes 3.2.4 et 3.2.5.

— IF /Condition/ THEN GOTO /N° de ligne/ ELSE GOTO /N° de ligne/ peut s'écrire indifféremment:

IF /Condition/ THEN /N° de ligne/ ELSE /N° de ligne/ et aussi:
IF /Condition/ GOTO /N° de ligne/ ELSE /N° de ligne/.

— Les IF et les ELSE imbriqués sont autorisés:

On peut écrire:
IF /Cond1/ THEN IF /Cond2/ THEN ...

Ou encore:
IF /Cond1/ THEN IF /Cond2/ THEN ... ELSE /Instruction/
IF /Cond1/ THEN ... ELSE IF /Cond2/... etc...

— Si l'un des numéros de ligne servant d'argument n'existe pas, une erreur est provoquée.

***Exemples***:

```
10 IF A>B AND A<C THEN PRINT "A est compris entre B et C"

10 IF A/2=INT(A/2) THEN ? "A est pair" ELSE ? "A est impair"

10 INPUT "VOULEZ-VOUS JOUER ";A$
20 IF A$<>"OUI" AND A$<>"NON" THEN 100 ELSE 200
30 END
100 PRINT "REPONDEZ PAR OUI OU NON"
110 GOTO 10
200 IF A$="NON" THEN 300
210 PRINT "BRAVO"
220 END
300 PRINT "DOMMAGE"
310 END
```

A RETENIR

## ON GOSUB/GOTO

*Syntaxe*: ON X GOTO /N° de ligne/[,/N° de ligne/,(...)]
et ON X GOSUB /N° de ligne/[,/N° de ligne/,(...)]

*Application*: Permet le branchement calculé vers un numéro de ligne (GOTO) ou vers le début d'un sous-programme (GOSUB) en fonction du résultat d'une expression (X). Si X=1, le GOTO ou le GOSUB s'effectuera vers la première ligne de la liste, si X=2 vers le deuxième et ainsi de suite.

REMARQUES:

— Si X=0, ou est supérieur au nombre de lignes spécifié, le programme continuera à la ligne suivante.

— Une erreur est provoquée si X est négatif ou supérieur à 255.

— Si X provient d'un calcul et comporte une partie décimale, il sera arrondi à l'entier inférieur.

— Il existe un autre moyen pour effectuer des branchements calculés: En effet, le MSX-BASIC accepte parfaitement une syntaxe du type GOTO X*100+10, ou GOSUB 10000–X*10.

***Exemple***:

```
10 CLS
20 INPUT "VOULEZ-VOUS ENTREZ UNE CHAINE OU UN NOMBRE (1/2)";A
30 ON A GOTO 100,200
40 PRINT:PRINT "ENTREZ 1 OU 2 SVP"
50 GOTO 20
100 REM BRANCHEMENT 1
110 PRINT:INPUT "ENTREZ UNE CHAINE ";A$
120 PRINT:PRINT "CHAINE :";A$
130 PRINT:GOTO 20
200 REM BRANCHEMENT 2
210 PRINT:INPUT "ENTREZ UN NOMBRE ";A
220 PRINT:PRINT "NOMBRE :";A
230 PRINT:GOTO 20
```

### 3.3.4. Les branchements automatiques

MSX-BASIC possède un jeu d'instructions spécialisées dans le "déroutement". Associées à des fonctions spécifiques du langage ou à des touches du clavier, ces commandes s'utilisent pour effectuer un branchement automatique vers un sous-programme chaque fois que la touche est actionnée ou la fonction validée. Le branchement est prioritaire sur toutes les instructions du Basic et interrompt le déroulement du programme, d'où le nom de "déroutement".

Ce système très puissant autorise une amélioration du dialogue avec l'utilisateur: La fonction ON KEY par exemple permet de saisir n'importe

quelle touche de fonction à n'importe quel moment du programme. Dans un jeu, ON SPRITE détecte la collision de mobiles à l'écran et ON STRIG contrôle en permanence les contacts du manche à balai !
Pour valider un déroutement, il faut déclarer en début de programme la touche ou la fonction qui doit servir à réaliser le branchement. On dispose pour ce faire de cinq déclarations réservées :

- ON INTERVAL... GOSUB ?
- ON STOP GOSUB (CTRL + STOP)
- ON KEY GOSUB (une touche quelconque pressée) (§ 4.2.2)
- ON STRIG GOSUB (Trig ou espace pressé) (§ 4.4)
- ON SPRITE GOSUB (collision de sprite) (§ 5.3)

Le déroutement proprement dit s'effectue ensuite à l'aide des instructions suivantes :

- INTERVAL ON/OFF/STOP
- STOP ON/OFF/STOP
- KEY(N) ON/OFF/STOP (§ 4.2.2)
- STRIG(N) ON/OFF/STOP (§ 4.4)
- SPRITE ON/OFF/STOP (§ 5.3)

Nous n'étudierons dans ce chapitre que les deux déclarations d'usage général (ON INTERVAL et ON STOP), les trois dernières s'utilisent avec des périphériques ou en mode graphique et seront donc présentées aux paragraphes correspondant.

REMARQUE : Le système de traitement des erreurs du MSX est très proche dans son principe du déroutement, mais effectue un branchement avec GOTO au lieu de GOSUB (voir ON ERROR GOTO, § 3.3.6).

### INTERVAL ON/OFF/STOP

*Syntaxes* : INTERVAL ON
INTERVAL OFF
INTERVAL STOP

*Applications* :

- INTERVAL ON active le déroutement déclaré par l'instruction ON INTERVAL GOSUB : Après exécution, le Basic effectue automatiquement un contrôle du temps écoulé avant exécution d'une nouvelle commande. Si cette période correspond à l'intervalle spécifié dans ON INTERVAL GOSUB, le déroutement vers le sous-programme est validé.
- INTERVAL OFF désactive ce déroutement : C'est le mode par défaut à la mise sous tension.
- INTERVAL STOP saisit et mémorise le déroutement spécifié dans ON INTERVAL GOSUB mais inhibe son fonctionnement jusqu'à rencontre d'un INTERVAL ON.

*Voir aussi* : ON INTERVAL GOSUB.

## ON INTERVAL GOSUB

*Syntaxe*: ON INTERVAL = /Temps écoulé/ GOSUB /N° de ligne/

*Application*: La rencontre de cette instruction provoque un branchement automatique tous les /Temps écoulé/ à un sous-programme débutant à /N° de ligne/. Pour être exécuté, le déroutement doit être validé par INTERVAL ON.

REMARQUES:

— /Temps écoulé/=50 correspond à une durée approximative de 1 seconde.

— Chaque fois qu'un déroutement est validé, un INTERVAL STOP est automatiquement exécuté. En retour de sous-programme, un INTERVAL ON supprime l'inhibition des déroutements.

— Les déroutements ne fonctionnent qu'en mode programme.

*Voir aussi*: INTERVAL ON/OFF/STOP

***Exemple***: Programme affichant toutes les huit secondes le temps écoulé depuis le lancement de l'exécution:

```
10 TEMPS=0:REM INITIALISE COMPTEUR TEMPS ECOULE
15 REM
20 ON INTERVAL=400 GOSUB 100:REM FIXE L'INTERVALLE A 8 SECONDES
25 REM
30 INTERVAL ON:REM VALIDE LE DEROUTEMENT
40 CLS
60 PRINT "BONJOUR, CHER UTILISATEUR"
70 PRINT
80 INPUT "ON CONTINUE (OUI/NON) ";A$
90 IF A$="OUI" THEN 40 ELSE END
100 REM SOUS-PROGRAMME DE DEROUTEMENT
110 TEMPS=TEMPS+8
110 PRINT
120 PRINT  "IL  S'EST  ECOULE ";TEMPS;" SECONDES DEPUIS LE DEBUT DU
PROGRAMME"
130 PRINT
140 RETURN
```

## ON STOP GOSUB

*Syntaxe*: ON STOP GOSUB /N° de ligne/

*Application*: L'action combinée de CONTROL et STOP interrompt le déroulement d'un programme et retourne au mode direct. Il est possible de dérouter l'action de ces touches pour effectuer un branchement vers un sous-programme avec ON STOP GOSUB. Comme pour les autres types de déroutements, l'instruction STOP ON valide cette déclaration.

REMARQUES :

— Chaque fois qu'un déroutement est validé, un STOP STOP est automatiquement exécuté. En retour de sous-programme, un STOP ON supprime l'inhibition des déroutements.

— ON STOP GOSUB est très utile pour protéger un programme contre une fausse manœuvre (voir exemple).

— Les déroutements ne fonctionnent qu'en mode programme.

*Voir aussi* : STOP ON/OFF/STOP.

***Exemple*** : Inhibition partielle de CONTROL/STOP :

```
10 ON STOP GOSUB 100
20 STOP ON
30 INPUT "ENTREZ VOTRE NOM ";NOM$
40 PRINT
50 PRINT "VOUS VOUS APPELEZ ";NOM$
60 PRINT
70 GOTO 30
100 REM DEROUTEMENT
110 INPUT "CONFIRMEZ LA SORTIE PAR OUI ";A$
120 IF A$="OUI" THEN END ELSE RETURN
```

STOP interrompu par CONTROL.STOP renvoie à la routine

Seule l'action de CONTROL/STOP autorise la sortie de ce programme. Le sous-programme débutant à la ligne 100 demande confirmation avant de valider la commande.

## STOP ON/OFF/STOP

*Syntaxes* : STOP ON
STOP OFF
STOP STOP

*Applications* :

— STOP ON valide le déroutement par la déclaration ON STOP GOSUB.

— STOP OFF supprime la prise en compte de ON STOP GOSUB. L'action combinée de CONTROL/STOP interrompt le déroulement du programme et retourne au mode direct.

— STOP STOP mémorise l'action de CONTROL/STOP mais ne valide le déroutement qu'après rencontre de STOP ON (voir exemple).

*Voir aussi* : ON STOP GOSUB.

***Exemple*** : Il est possible de ne valider le CONTROL/STOP qu'après exécution d'une série d'instructions. Reprenons l'exemple précédent :

```
10 ON STOP GOSUB 100
20 STOP STOP
30 INPUT "ENTREZ VOTRE NOM ";NOM$
40 PRINT
50 PRINT "VOUS VOUS APPELEZ ";NOM$
60 PRINT
70 STOP ON
80 GOTO 20
100 REM DEROUTEMENT
110 INPUT "CONFIRMEZ LA SORTIE PAR OUI ";A$
120 IF A$="OUI" THEN END ELSE RETURN
```

L'action de CONTROL/STOP est inhibée jusqu'à saisie et affichage du nom. Le déroutement s'effectue ensuite normalement.

### 3.3.5. Les déclarations

On appelle déclaration une instruction du Basic qui ne donne pas lieu à une exécution immédiate mais dont l'emploi sert à imposer un format ou réaliser une affectation particulière pour des variables du Basic: Taille d'un tableau, type d'une variable, définition d'une fonction.

REMARQUE : On regroupe en général toutes les déclarations au début d'un programme pour en améliorer la clarté et faciliter les modifications éventuelles.

## DEF FN

*Syntaxe* : DEF FN/Nom de fonction/ [(/Liste variables/)] = /Fonction/

*Application* : Cette instruction permet au programmeur de définir ses propres fonctions, qui seraient éventuellement absentes du Basic.

— /Nom de fonction/ suit les règles d'appellation des variables ; il servira par la suite à identifier la fonction.

— La liste de variables qui suit le nom de la fonction est optionnelle. Elle correspond aux paramètres qui doivent être stipulés dans FN lors de son appel : Il s'agit des variables sur lesquelles la fonction effectue le traitement. Chaque variable doit être séparée de la suivante par une virgule.

— /Fonction/ est une expression définissant la fonction. Si les noms de variables utilisés dans cette expression correspondent à /Liste de variables/, leur valeur est passée par le programme lors de l'appel de la fonction.

Au cas où la liste est omise, les variables de l'expression sont directement liées à celles du programme.

— Pour l'utilisation de la fonction une fois celle-ci définie, voir aussi FN au chapitre traitant des fonctions (§ 3.4.1).

REMARQUES :

— Une fonction peut travailler indifféremment sur des chaînes ou des nombres.

— L'expression servant à définir une fonction ne peut dépasser la longueur d'une ligne de programme.

— Le nom de variable qui suit FN force le type de la fonction. Il doit correspondre à l'expression la définissant, faute de quoi une erreur est provoquée.

— DEF FN ne fonctionne qu'en mode programme.

*Voir aussi* : FN.

***Exemples*** :

| | |
|---|---|
| `DEF FNTA (X)=SIN(X)/COS(X)` | La fonction FNTA calcule la tangente de l'angle X. |
| `DEF FNAD (X,Y)=X^2+Y^2` | La fonction FNAD additionne les carrés de X et Y. |
| `DEF FNCO$ (A$,B$,C$)=A$+B$+C$` | La fonction FNCO$ concatène les chaînes A$, B$ et C$. |

Dans les trois cas qui précèdent, les fonctions effectuent le traitement sur des variables passées par le programme principal sous forme d'une liste de paramètres. Les noms de variables utilisés sont fictifs et peuvent être remplacés par d'autres lors de l'appel de la fonction (voir FN). Il est également possible de définir une fonction travaillant sur les valeurs courantes des variables, sans passage de paramètres :

DEF FNAD=A+B additionne A et B en utilisant les valeurs de ces deux variables au moment de l'appel de la fonction.

**DEFDBL**

*Syntaxe* : DEFDBL /Suite de lettres/

*Application* : DEFDBL permet de forcer le type des variables dont le nom commence par une des lettres spécifiées en double précision. On peut entrer une liste de lettres en argument de DEFDBL en les séparant par des virgules.

Si une suite alphabétique doit être spécifiée en double précision, il suffit de rentrer la première et la dernière lettre en les séparant par des tirets.

REMARQUE : Les identificateurs !, % et $ ont priorité sur DEFDBL : On

peut par leur intermédiaire forcer le type d'une variable préalablement définie.

*Voir aussi*: DEFINT, DEFSGN, DEFSTR.

***Exemples***:

DEFDBL A-G Toutes les variables dont le nom débute par une lettre allant de A à G sont en double précision, sauf spécification contraire avec ! ou %.

DEFDBL A,H Les variables dont les noms débutent par A et H sont en double précision.

**DEFINT**

*Syntaxe*: DEFINT /Suite de lettres/

*Application*: Comme DEFDBL, DEFINT permet de forcer comme entières les variables dont le nom commence par une des lettres spécifiées. Les règles de syntaxe sont identiques pour DEFINT et DEFDBL.

REMARQUE : La déclaration systématique de toutes les variables utilisées dans les boucles comme entières améliore sensiblement les performances d'un programme.

*Voir aussi*: DEFDBL, DEFSGN, DEFSTR.

**DEFSGN**

*Syntaxe*: DEFSGN /Suite de lettres/

*Application*: Toutes les variables dont la première lettre appartient à /Suite de lettres/ sont en simple précision (voir DEFDBL pour la syntaxe précise).

*Voir aussi*: DEFDBL, DEFINT, DEFSTR.

**DEFSTR**

*Syntaxe*: DEFSTR /Suite de lettres/

*Application*: Il est possible de se passer du suffixe $ pour déclarer une variable chaîne en choisissant une gamme de lettres réservées par DEFSTR.

REMARQUE: On applique les mêmes règles de syntaxe que pour DEFDBL.

*Voir aussi*: DEFINT, DEFSGN, DEFDBL.

**DEFUSR**

*Syntaxe* : DEFUSR[A]=/Adresse/

*Application* : Il est parfois nécessaire d'inclure des routines en langage-machine à l'intérieur d'un programme en Basic interprété. Pour ce faire, il faut ranger en mémoire la suite d'octets correspondant au programme assembleur à une adresse connue du basic. La déclaration DEFUSR permet de stocker l'adresse de 10 routines différentes numérotées de 0 à 9. La fonction USR sert ensuite à lancer l'exécution du programme (voir § 3.4.1). La variable /Adresse/ définit l'adresse de début de la routine et doit être un entier compris entre 0 et 65535. [A] sert à numéroter la fonction et doit être compris entre 0 et 9.

REMARQUE : Si A est omis, DEFUSR0 est pris par défaut.

*Voir aussi* : USR.

**DIM**

*Syntaxe* : DIM /Tableau 1/(/Dim.1/[/,Dim.2/](...)[,/Tableau 2/(...)]

*Application* : Pour définir les dimensions des tableaux avant leur utilisation. On a vu dans la première partie (§ 2.1.3) qu'il est possible d'utiliser des tableaux comportant jusqu'à dix éléments par dimension sans avoir à les déclarer : Le fait de nommer un élément de tableau le dimensionne automatiquement.

Dans le cas où plus de dix éléments doivent être utilisés, l'instruction DIM sert à spécifier la taille d'un ou plusieurs tableaux avant utilisation :

— /Tableau 1/, /Tableau 2/ sont des noms de variables standard qui serviront ensuite à caractériser les tableaux.

— /Dim.1/, /Dim.2/ sont des entiers affectés à chacune des dimensions du tableau pour en définir la taille.

L'emploi des éléments d'un tableau se fait ensuite à l'aide de variables indicées, dont la syntaxe est fonction du nombre de dimensions choisi, et dont les indices sont liés aux valeurs de ces dimensions. Leur emploi étant assez complexe, voyez aussi les exemples et programmes présentés plus loin pour les méthodes pratiques à mettre en œuvre.

REMARQUES :

— Le MSX-BASIC accepte jusqu'à 255 dimensions pour un tableau, avec un nombre quelconque d'éléments par dimension : C'est la taille mémoire disponible qui impose les limites.

— Le premier élément d'un tableau a toujours pour indice 0 : DIM A(12) définit donc un tableau à une dimension de 13 éléments, dont le premier est A(0) et le dernier A(12).

— Tous les types de variables du MSX-BASIC peuvent être organisés en tableaux : Entiers, simple et double précision, chaînes.

— Une fois un tableau dimensionné, toute tentative de la redimensionner se soldera par un message d'erreur. Il faut soit remettre tous les éléments du tableau à 0, soit utiliser CLEAR, RUN ou ERASE qui effacent les variables de la mémoire en bloc ou sélectivement.

— On peut dimensionner plusieurs tableaux dans une instruction DIM, en séparant leurs noms par des virgules :

```
DIM A(12),B(11),C$(10,30)
```

sont des syntaxes correctes.

```
DIM A(5,5,15),A%(20,20)
```

— Rappelons qu'en ce qui concerne les tableaux de chaînes, l'élément est la chaîne, et non le caractère comme pour certains autres Basic ; la conséquence pratique en est que la longueur des chaînes n'a pas besoin d'être définie à l'avance, elle peut librement varier en cours de programme.

*Voir aussi* : ERASE, CLEAR.

***Exemples*** :

**Les tableaux à une dimension** : DIM A(L)

DIM A$(15) déclare un tableau contenant 16 chaînes dont la longueur peut varier entre 0 et 255. On peut mémoriser une série de noms dans un tableau de chaînes :

```
 5 CLS
10 DIM NOM$(15):REM DIMENSIONNE NOM$
20 FOR I=0 TO 15
30 INPUT "ENTREZ UN NOM ";NOM$(I):REM REMPLIT LE TABLEAU
40 PRINT
50 NEXT I
60 PRINT "LISTE DES NOMS :":PRINT:REM AFFICHE SON CONTENU
70 FOR I=0 TO 15
80 PRINT NOM$(I)
90 NEXT I
```

**Les tableaux à deux dimensions** : DIM A(L,P)

On a vu au paragraphe 2.1.3 qu'un tableau à deux dimensions peut être comparé à un livre, où P représente un numéro de page et L un numéro de ligne dans la page. On peut adresser chaque élément du tableau avec :

```
A(L,P)
```

Dans l'exemple qui suit, on associe à chaque élément son "numéro de ligne", puis on affiche le contenu du tableau :

```
10 CLS
20 DIM A(3,3)
30 FOR P=0 TO 3
40  FOR L=0 TO 3
50   A(L,P)=L
60   PRINT "A(";L;",";P;")=";A(L,P)
70  NEXT L
80 NEXT P
```

**Les tableaux à trois dimensions :** DIM A(L,P,V)

Si la première dimension représente un numéro de ligne et la deuxième un numéro de page, on peut par analogie considérer la troisième dimension comme un numéro de volume. On adresse chaque élément d'un tableau à trois dimensions avec :

```
A(L,P,V)
```

L'exemple précédent peut être aisément étendu à trois dimensions au lieu de deux :

```
10 CLS
20 DIM A(3,3,2)
30 FOR V=0 TO 2
40  FOR P=0 TO 3
50   FOR L=0 TO 3
60    A(L,P,V)=L
70     PRINT "A(";L;",";P;",";V;")=";A(L,P,V)
80    NEXT L
90   NEXT P
100 NEXT V
```

Les tableaux à plusieurs dimensions peuvent servir à créer des fichiers d'adresse :

```
 5 CLS
10 DIM NOM$(8,2):REM DIMENSIONNE NOM$
20 FOR I=0 TO 8
25 FOR J=0 TO 2
30 ON J+1 GOSUB 200,300,400
40 INPUT NOM$(I,J):REM REMPLIT LE TABLEAU
45 PRINT
50 NEXT J,I
60 PRINT "LISTE DES NOMS :":PRINT:REM AFFICHE SON CONTENU
70 FOR I=0 TO 8
75 FOR J=0 TO 2
80 PRINT NOM$(I,J);"  ";
90 NEXT J
100 PRINT
110 NEXT I
120 END
```

A retenir. erase A$(X), C(X), ...

```
200 REM SAISIE DU NOM
210 PRINT "ENTREZ UN NOM ";
220 RETURN
300 REM SAISIE DU PRENOM
310 PRINT "ENTREZ UN PRENOM ";
320 RETURN
400 REM SAISIE DU TELEPHONE
410 PRINT "ENTREZ UN TELEPHONE ";
420 RETURN
```

**ERASE**

*Syntaxe*: ERASE /Tableau 1/( )[,/Tableau 2/( )](...)

*Application*: ERASE permet d'éliminer un ou plusieurs tableaux de la mémoire, sans toucher aux autres variables du Basic. Après effacement par cette instruction, un tableau peut être redimensionné sans provoquer une erreur.

*Voir aussi*: DIM, CLEAR

***Exemple***: `ERASE  A$(),NOM$(),C()` : efface de la mémoire les tableaux A$, NOM$ et C.

### 3.3.6. Le traitement des erreurs

Chaque fois qu'une instruction ou un calcul ne peut être exécuté par l'interpréteur du MSX-BASIC, une erreur est provoquée, le déroulement du programme arrêté et un message correspondant au type de l'erreur affiché: Une division par 0, une faute de syntaxe, la tentative d'affecter à une variable numérique une constante chaîne sont des erreurs classiques en programmation. Trois commandes du Basic autorisent le branchement à une routine de traitement et l'analyse du code de chaque erreur. On les utilise pour protéger un programme des fausses manœuvres et pour personnaliser les messages d'erreur en fonction des cas particuliers rencontrés dans un programme (voir exemple).

REMARQUES :

— Le système de traitement des erreurs du MSX-BASIC est très proche dans son principe des branchements automatiques vu au paragraphe 3.3.4: La déclaration ON ERROR GOTO provoque en cas d'erreur un déroutement prioritaire vers la ligne de programme spécifiée.

— Les codes et messages d'erreur ainsi que leur signification sont répertoriés à l'appendice 2.

a retenir

**ON ERROR GOTO**

*Syntaxe*: ON ERROR GOTO /N° de ligne/

*Application*: Active le déroutement vers la ligne spécifiée chaque fois qu'une erreur est rencontrée durant le déroulement du programme.

REMARQUES :

— Si /N° de ligne/ n'existe pas, le programme est interrompu et un message "Undefined line number" est affiché.

— ON ERROR GOTO 0 supprime le traitement des erreurs par ON ERROR. Si cette commande est rencontrée dans la routine de traitement des erreurs démarrant à /N° de ligne/, le programme s'interrompt et le message correspondant à l'erreur traitée est affiché.

— Si une erreur survient à l'intérieur de la routine de traitement des erreurs, l'erreur est saisie normalement et le message correspondant affiché.

— En cas de déroutement, deux variables réservées du Basic renvoient des informations sur l'erreur en cours :

**ERL** renvoie le numéro de la ligne où l'erreur s'est produite.

**ERR** renvoie le code de l'erreur.

*Voir aussi*: ERROR, RESUME.

***Exemple***: Affichage d'un message d'erreur en français :

ok

```
10 CLS
20 ON ERROR GOTO 1000
30 INPUT "ENTREZ UN NOMBRE DIFFERENT DE 0 ";A
35 B=1/A
40 PRINT:PRINT "INVERSE DE ";A;"=";B
50 PRINT:GOTO 30
1000 REM ROUTINE DE TRAITEMENT DES ERREURS
1010 IF ERR=11 THEN PRINT:PRINT"IL FAUT ENTRER UN NOMBRE
DIFFERENT DE 0 !"
1020 PRINT
1030 GOTO 30
```

Dans la routine de déroutement, un message s'affiche si le code de l'erreur ayant provoqué l'aiguillage correspond à la tentative d'entrer pour A une constante chaîne. L'exécution est reprise à la ligne de saisie.

**RESUME**

*Syntaxe*: RESUME [A]

*Application*: Après déroutement par ON ERROR, il est possible de reprendre avec RESUME le déroulement normal du programme :

— Si A est omis ou égal à 0, l'exécution reprend à la ligne ayant causé l'erreur.

— Si A est égal à un numéro de ligne, l'exécution reprend à ce numéro.

— Enfin, RESUME NEXT permet la reprise du programme à la ligne suivant celle où l'erreur s'est produite.

REMARQUE : Un RESUME qui n'est pas inséré dans une routine de déroutement provoque une erreur.

*Voir aussi*: ON ERROR, ERROR.

***Exemple*** : Reprenons l'exemple précédent :

```
10 CLS
20 ON ERROR GOTO 1000
30 INPUT "ENTREZ UN NOMBRE DIFFERENT DE 0 ";A
35 B=1/A
40 PRINT:PRINT "INVERSE DE ";A;"=";B
50 PRINT:GOTO 30
1000 REM ROUTINE DE TRAITEMENT DES ERREURS
1005 IF ERR<>11 THEN END
1010 PRINT :PRINT "IL FAUT ENTRER UN NOMBRE DIFFERENT DE 0 !"
1020 PRINT
1030 PRINT "Code de l'erreur:";ERR
1035 PRINT
1040 PRINT "Ligne de l'erreur:";ERL
1050 PRINT
1060 RESUME 30
```

Cette fois, l'exécution du programme est reprise à la ligne 30 uniquement si le code renvoyé par ERR correspond à une erreur de saisie. Les informations contenues dans ERL et ERR sont affichées sous le message d'erreur. Dans le cas contraire, le déroulement du programme est interrompu.

## ERROR

*Syntaxe*: ERROR /Valeur entière/

*Applications*: ERROR sert principalement à définir des codes d'erreur personnalisés.

— Si /Valeur entière/ correspond à un code d'erreur du Basic, ERROR simule cette erreur, avec interruption du programme et affichage d'un message.

— Si /Valeur entière/ est supérieure ou égale à 60, l'erreur provoquée ne peut être traitée et un message "Unprintable error" est affiché.

Si un déroutement par ON ERROR est déclaré en début de programme, on peut gérer cette erreur comme n'importe quelle autre et créer des messages en fonction du code choisi.

REMARQUE : /Valeur entière/ doit être comprise entre 0 et 255.

*Voir aussi* : ON ERROR, RESUME.

***Exemple*** : Création d'une erreur de code 200 :

```
10 CLS
20 ON ERROR GOTO 1000
30 PRINT:INPUT "ENTREZ UN NOM ";NOM$
40 PRINT:PRINT NOM$
50 PRINT:INPUT "ON CONTINUE (OUI/NON) ";A$
60 IF A$<>"OUI" AND A$<>"NON" THEN ERROR 200
70 IF A$="OUI" THEN GOTO 30 ELSE END
1000 REM ROUTINE DE DEROUTEMENT
1010 IF ERR=200 THEN PRINT:PRINT"ENTRER OUI OU NON":RESUME 50
1020 PRINT:PRINT "ERREUR INCONNUE"
```

## 3.4. LES FONCTIONS DU MSX

A la différence d'une instruction, une fonction s'utilise toujours en conjonction avec une autre commande du Basic, en général PRINT ou LET. Par exemple : LET A=VAL(A$).

Les syntaxes données pour les fonctions suivantes doivent en conséquence être considérées comme incomplètes par le lecteur. Comme pour les Instructions, on peut regrouper les Fonctions selon trois catégories :

| **USAGE GENERAL** | | **CALCUL** | **TRAITEMENT CHAINES** |
|---|---|---|---|
| BASE | INPUT$ | ABS | ASC |
| BIN$ | INT | ATN | CHR$ |
| CDBL | LPOS | COS | INSTR |
| CINT | OCT$ | EXP | LEFT$ |
| CSNG | PEEK | LOG | LEN |
| FIX | RND | SGN | MID$ |
| FN | SPC | SIN | RIGHT$ |
| FRE | TAB | SQR | SPACE$ |
| HEX$ | TIME | TAN | STR$ |
| INKEY$ | USR | | STRING$ |
| INP | VARPTR | | |

### 3.4.1. Les fonctions d'usage général

**BASE**

*Syntaxe* : BASE (/N/)

*Application* : BASE renvoie l'adresse de début en mémoire des zones de stockage des données servant à l'affichage. /N/ doit être un entier compris entre 0 et 18 :

Mode texte 40 colonnes :

| | |
|---|---|
| N=0 | Buffer texte |
| N=2 | Générateur de caractères |

Mode texte 32 colonnes :

| | |
|---|---|
| N=5 | Buffer texte |
| N=6 | Table des couleurs |
| N=7 | Générateur de caractères |
| N=8 | Affectation des sprites |
| N=9 | Dessin des sprites. |

Mode haute résolution (256 × 192) :

| | |
|---|---|
| N=10 | Buffer texte |
| N=11 | Table des couleurs |
| N=12 | Générateur de caractères |
| N=13 | Affectation des sprites |
| N=14 | Dessin des sprites. |

Mode basse résolution (64 × 48) :

| | |
|---|---|
| N=15 | Buffer texte |
| N=16 | Générateur de caractères |
| N=17 | Affectation des sprites |
| N=18 | Dessin des sprites. |

REMARQUE : La fonction BASE ne s'utilise que dans les programmes en langage-machine, et ne concerne pas les utilisateurs du Basic.

**BIN$**

*Syntaxe* : BIN$(N)

*Application* : Retourne une chaîne correspondant à l'équivalent binaire du nombre décimal N.

REMARQUES :

— N doit être compris entre –32768 et 65535. Si N est négatif, BIN$ effectue le complément à 2 de N avant la conversion c'est-à-dire BIN$(65536–N).

— BIN$ est le complément de &B (voir § 3.2).

*Voir aussi* : HEX$, OCT$ et BAL.

***Exemple*** :

```
10 REM CONVERSION DECIMAL-BINAIRE
20 CLS
30 ?:INPUT "ENTREZ UN NOMBRE DECIMAL ";A
40 A$=BIN$(A)
50 ?:?"EQUIVALENT BINAIRE: ";A$
60 GOTO 30
```

## CDBL

*Syntaxe* : CDBL(X)

*Application* : Convertit le nombre X en double précision.

REMARQUE : Le transfert de X dans une variable en double précision X# est équivalent à CDBL: 10 X#=X:X=X#.

*Voir aussi* : CINT, CSNG.

## CINT

*Syntaxe* : CINT(X)

*Application* : La fonction CINT convertit X en nombre entier, en supprimant la partie décimale.

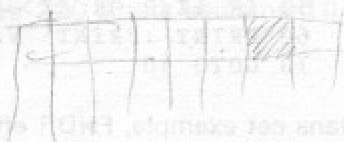
REMARQUES :

— X doit être compris entre −32768 et 32767 sinon une erreur est provoquée.

— CINT(A) correspond à la séquence 10 A%=A:A=A%.

*Voir aussi* : INT, FIX.

***Exemple*** :



```
CINT(-3.7)  donne  -4
CINT (3.7)  donne   3
```

## CSNG

*Syntaxe* : CSNG(X)

*Application* : Conversion de X en simple précision.

REMARQUE : CSNG(A) correspond à la séquence 10 A!=A:A=A!.

*Voir aussi* : CINT, CDBL.

## FIX

*Syntaxe* : FIX(X)

*Application*: Comme CINT, FIX renvoie la partie entière de X, c'est-à-dire la partie de X située à gauche du point décimal.

REMARQUE: FIX(X) est équivalent à SGN(X)*INT(ABS(X)). Un nombre négatif est converti en positif avant traitement, puis reconverti en négatif.

*Voir aussi*: CINT, INT.

***Exemples***:

```
FIX(-3.7)  donne -3
FIX (3.7)  donne  3
```

**FN**

*Syntaxe*: FN/Nom de la fonction/[(/Liste de paramètres/)]

*Application*: Une fonction préalablement définie avec DEF FN (cf. cette instruction), s'utilise ensuite comme n'importe quelle autre fonction standard (SIN ou TAN par exemple) grâce à la fonction FN.

REMARQUE: /Nom de la fonction/ et /Liste de paramètres/ suivent les mêmes règles que DEF FN.

*Voir aussi*: DEF FN.

***Exemples***:

```
10 REM CONVERSION DEGRES-RADIANS
30 DEF FNDR(X)=(3.14159*X)/180
35 CLS
40 PRINT :INPUT "ENTREZ UNE VALEUR EN DEGRES ";A%
50 IF A%<0 OR A%>360 THEN GOTO 35
60 PRINT :PRINT "VALEUR EN RADIANS :";FNDR(A%)
70 GOTO 40
```

Dans cet exemple, FNDR effectue la conversion sur A%: Le paramètre inclus entre parenthèses remplace X dans l'expression DEF FN pour le calcul.

Il est possible de passer plusieurs paramètres par FN:

```
10 REM CONCATENATION DE CHAINES
20 DEF FNAD$(A$,B$,C$)="NOM: "+A$+"/PRENOM: "+B$+"/TEL: "+C$
30 CLS
40 PRINT :INPUT"ENTREZ VOTRE NOM ";NOM$
50 PRINT :INPUT"ENTREZ VOTRE PRENOM ";PRENOM$
60 PRINT :INPUT"ENTREZ VOTRE TELEPHONE ";TEL$
70 CLS
80 PRINT "FICHE IDENTITE: "
90 PRINT :PRINT FNAD$(NOM$,PRENOM$,TEL$)
100 END
```

Si aucun paramètre n'est inséré lors de la définition de la fonction, les valeurs courantes des variables au moment de l'appel sont utilisées :

```
10 REM CONCATENATION DE CHAINES
20 DEF FNAD$="NOM: "+A$+"/PRENOM: "+B$+"/TEL: "+C$
30 CLS
40 PRINT :INPUT"ENTREZ VOTRE NOM ";A$
50 PRINT :INPUT"ENTREZ VOTRE PRENOM ";B$
60 PRINT :INPUT"ENTREZ VOTRE TELEPHONE ";C$
70 CLS
80 PRINT "FICHE IDENTITE: "
90 PRINT :PRINT FNAD$
100 END
```

## FRE

*Syntaxes* : FRE(O)
FRE("")

*Applications* :

— Dans sa première syntaxe, cette fonction sert à connaître le nombre d'octets encore inoccupés en mémoire, pendant l'écriture d'un programme par exemple.

— Dans sa deuxième syntaxe, la fonction retourne aussi la quantité de mémoire disponible pour les chaînes de caractères. Il est possible de redéfinir cette quantité avec l'instruction CLEAR.

REMARQUE : A la différence de certains Basic, la taille de l'espace chaîne utilisé par le MSX ne varie pas au cours d'un programme. Il est donc indispensable de définir dans une application la quantité de mémoire qu'utiliseront les chaînes, et de rentrer cette valeur dans CLEAR en tête de programme (la valeur par défaut réserve 200 octets seulement !).

*Voir aussi* : CLEAR, ERASE.

***Exemple*** :

```
10000 IF FRE(0)<200 THEN PRINT "PLUS DE MEMOIRE !"
10010 RETURN
```

Ce sous-programme peut être ajouté à un programme "gourmand" pour vous prévenir quand il n'y a presque plus de mémoire libre.

## HEX$

*Syntaxe* : HEX$(N)

*Application* : Retourne une chaîne correspondant à l'équivalent hexadécimal du nombre N.

REMARQUES :

— N doit être compris entre –32768 et 65535.

— Si N est négatif, HEX$ le convertit en son complément à 2 avant d'effectuer la conversion : HEX$(65536–N).

— HEX$ est la fonction inverse de &H (voir § 3.2.1).

— Lorsqu'on raisonne sur des octets, il est très pratique d'utiliser le système hexadécimal : Si l'on sépare un octet de huit bits en deux "mots" de 4 bits, chacun d'entre eux peut s'écrire sur un seul chiffre hexadécimal puisqu'il ne peut prendre que 16 valeurs différentes (2 puissance 4 voir § 3.2) : Un octet, s'exprimera donc toujours par un nombre hexadécimal à deux chiffres.

En pratique, il est très facile de diviser un octet en deux groupes de 4 bits : Dans le nombre binaire 0111 1111 (127), le groupe de droite vaut 15, donc F, et le groupe de gauche vaut 7, qui reste 7 en base 16. Cet octet s'écrira donc "7F" en hexadécimal.

*Voir aussi* : VAL, OCT$, BIN$.

***Exemples*** :

PRINT HEX$(65535) nous donne FFFF :

```
10 REM CONVERSION DECIMAL-BINAIRE-HEXADECIMAL
20 CLS
30 ?:INPUT"ENTRER UN NOMBRE DECIMAL ";A
40 ?:?"DECIMAL: ";A
50 ?:?"BINAIRE: ";BIN$(A)
60 ?:?"HEXADECIMAL: ";HEX$(A)
70 GOTO 30
```

## INKEY$

*Syntaxe* : X$=INKEY$

*Application* : La fonction de saisie INKEY$ scrute le clavier de façon fugitive : L'exécution du programme continue après que la fonction ait été rencontrée, qu'une touche ait été actionnée ou non à ce moment. Le caractère de la touche pressée est stocké dans X$, qui sera une chaîne vide si aucune touche n'est utilisée. La fonction doit donc être incluse dans des boucles pour lire efficacement le clavier. C'est la solution idéale pour les jeux, où le clavier doit être constamment surveillé sans pour cela empêcher d'autres actions de s'exécuter simultanément.

REMARQUES :

— INKEY$ ne saisit qu'un seul caractère à la fois dans X$.

— La valeur de X$ est passée directement au programme sans affichage à l'écran.

*Voir aussi*: INPUT$.

***Exemple***:

```
10 REM SAISIE DE TEXTE AU KILOMETRE
15 CLS
20 X$=INKEY$
25 IF X$=CHR$(13) THEN PRINT:GOTO 20
30 PRINT X$;
40 GOTO 20
```

**Explications:**

— Si aucune touche n'est actionnée, le déroulement du programme se poursuit, la ligne 30 imprime une chaîne vide et la ligne 40 retourne à la saisie.

— Si la touche RETURN est pressée, le caractère de code 13 est saisie et le programme passe à la ligne suivante (voir codes de contrôle § 1.4.3 et CHR$ § 3.4.3).

— Le caractère correspondant à toute autre touche est directement affiché à l'écran comme sur le papier d'une machine à écrire.

## INP

*Syntaxe*: INP(/N° de port/)

*Application*: INP retourne l'octet lu sur le port du Z 80 spécifié.

REMARQUES:

— /N° de port/ doit être compris entre 0 et 255.

— INP est la fonction complémentaire de l'instruction OUT.

*Voir aussi*: OUT.

## INPUT$

*Syntaxe*: INPUT$(X)

*Application*: Lorsque l'instruction X$=INPUT$(X) est rencontrée dans un programme, l'ordinateur attend l'entrée de X caractères à partir du clavier; Dès que cela est fait, la variable donnée comme argument X$ contiendra la chaîne correspondant aux touches actionnées; Le programme continue ensuite son exécution.

REMARQUES :

— A la différence de INPUT, n'importe quel caractère du clavier peut être saisi, excepté CONTROL/STOP qui interrompt le programme.

— La chaîne est directement transmise au programme sans affichage à l'écran.

— N doit être un entier compris entre 1 et 255.

*Voir aussi* : INKEY$.

***Exemple*** : Saisie avec conversion automatique en majuscules :

```
100 REM CONVERSION EN MAJUSCULES
110 CLS
120 ?"ENTREZ VOTRE NOM SUIVI DE [RETURN] : ";
130 GOSUB 300
140 PRINT CHR$(X);
150 IF X<>13 THEN 130
160 END
300 REM SAISIE ET CONVERSION
310 X$=INPUT$(1)
315 X=ASC(X$)
320 IF X=13 THEN RETURN
330 IF X>96 THEN X=X-32
340 RETURN
```

**Explications :**

— 120,130 : La saisie du nom s'effectue dans le sous-programme de ligne 300, caractère par caractère.

— 315 : La ligne 315 charge X avec le code ASCII de X$ (voir ASC § 3.4.3).

— 320 : Si le code de la touche pressée correspond à celui de la touche [RETURN] on retourne directement au programme principal.

— 330 : Si le code de la touche est supérieur à 96, le caractère est en minuscule, et doit être converti.

— 340 : Retour au programme principal. Le caractère correspondant à X est affiché (voir CHR$ § 3.4.3) et la saisie reprend si X n'est pas la code de [RETURN].

## INT

*Syntaxe* : INT(X)

*Application* : Retourne le plus grand nombre entier inférieur ou égal à X.

*Voir aussi* : CINT, FIX.

***Exemples*:**

```
INT (3.6)  donne  3
INT (-3.2)  donne -4
```

## LPOS

*Syntaxe*: LPOS(X)

*Application*: Retourne la position de la tête d'impression dans le buffer de l'imprimante, ce qui ne correspond pas forcément à sa position horizontale sur la largeur de la feuille de papier: Il s'agit plutôt de l'emplacement ou aura lieu la prochaine impression.

REMARQUE: X est un argument fictif qui peut prendre n'importe quelle valeur.

*Voir aussi*: POS, LPRINT.

## PCT$

*Syntaxe*: OCT$(N)

*Application*: Retourne une chaîne correspondant à l'équivalent octal du nombre décimal N.

REMARQUES:

— N doit être compris entre −32768 et 65535. Si N est négatif, OCT$ effectue le complément à 2 de N avant la conversion c'est-à-dire OCT$(65536-N).

— OCT$ est le complément de &O (voir § 3.2).

*Voir aussi*: HEX$, BIN$ et VAL.

***Exemple*:**

```
10 REM CONVERSION DECIMAL-OCTAL
20 CLS
30 ?:INPUT "ENTREZ UN NOMBRE DECIMAL ";A
40 A$=OCT$(A)
50 ?:?"EQUIVALENT OCTAL: ";A$
60 GOTO 30
```

## PEEK

*Syntaxe*: PEEK(I)

*Application*: Donne la valeur de l'octet contenu dans l'adresse I. I doit être compris entre −32768 et 65535.

REMARQUE: PEEK est le complément de l'instruction POKE.

*Voir aussi*: POKE.

***Exemple***: Lecture d'une adresse stockée à l'adresse A:

Une adresse est stockée sur deux octets, octet de poids faible en premier (voir POKE § 3.3.2). Pour lire sa valeur, on écrit:

```
ADRESSE=PEEK(A)+256*PEEK(A+1)
```

## RND

*Syntaxe*: RND(N)

*Application*: Cette fonction donne des résultats différents selon la valeur de N:

— Si N>0, elle retourne à chaque appel un nombre pseudo-aléatoire compris entre 0 et 1 non inclus. La séquence résultante est dite pseudo-aléatoire, car la suite des nombres qu'elle génère est fixe.

— Si N=0 elle renvoie toujours le nombre précédent.

— Si N<0, le nombre fourni est fonction de la valeur de N, et toujours le même pour des valeurs égales de N.

REMARQUE: Chaque fois qu'un RUN est exécuté, la séquence repart du même point.

***Exemple***: Jeu de dés électronique:

```
10 CLS
20 ?:?"POUR JOUER, TAPEZ UNE TOUCHE "
30 X$=INPUT$(1)
40 ?:?INT((RND(1)*6)+1)
50 GOTO 30
```

## SPC

*Syntaxe*: SPC(X)

*Application*: Imprime X espaces à l'écran ou sur l'imprimante. X est automatiquement converti en entier et doit être compris entre 0 et 255.

REMARQUE: SPC ne s'utilise qu'avec PRINT et LPRINT.

*Voir aussi*: PRINT, LPRINT, TAB.

***Exemple***:

```
PRINT SPC(10)"JE M'APPELLE"SPC(5)"MSX"
```

à vérifier.

## TAB

*Syntaxe* : TAB(X)

*Application* : TAB autorise le positionnement horizontal du curseur sur n'importe laquelle des 40 positions standard d'écriture, relativement au bord gauche de l'écran (ou de la feuille sur l'imprimante). N est un entier positif, compris entre 0 et 255 représentant le numéro de colonne. Le bord gauche de l'écran à pour valeur 0, le bord droit 39. Si N>39, le curseur saute N = 40 lignes et se place à la colonne calculée par N MOD 40.

REMARQUE : TAB ne s'emploie qu'avec PRINT et LPRINT.

*Voir aussi* : PRINT, LPRINT, SPC.

***Exemple*** :

```
PRINT "CHAINE";TAB(15);"DE CARACTERES"
```

## TIME

*Syntaxe* : TIME

*Application* : A la mise sous tension de l'ordinateur, le système possède une horloge interne qui s'incrémente de 1 cinquante fois par seconde (sur les machines vendues en France). TIME renvoie la valeur de cette horloge sous la forme d'un entier non signé.

REMARQUE : Il est possible de réinitialiser l'horloge pour chronométrer un événement en écrivant TIME=0.

***Exemple*** : Calcul de la durée d'une boucle :

```
10 TIME=0
20 FOR I%=1 TO 10000
30 NEXT I%
40 TE=TIME
50 ?"Temps écoulé: ";TE/50;"secondes"
```

## USR

*Syntaxe* : USR [/N/](X)

*Application* : Effectue l'appel d'une routine assembleur dont l'adresse a été définie préalablement par DEFUSR avec passage de l'argument X. /N/ doit être compris entre 0 et 9 et correspond à la valeur spécifiée pour la routine dans DEFUSR.

REMARQUE : Si /N/ est omis, la routine DEFUSR0 est prise par défaut.

*Voir aussi* : DEFUSR.

## VARPTR

*Syntaxes* : VARPTR (/Variable/)
VARPTR (#/N° de fichier/)

*Application* :

— Dans sa première syntaxe, VARPTR renvoie l'adresse de début en mémoire de la variable spécifiée. Tout type de variable peut être utilisé, numérique indicé ou chaîne.

— Dans la deuxième syntaxe, c'est l'adresse du bloc de contrôle du fichier spécifié qui est retournée par VARPTR. Ce bloc est une zone mémoire contenant toutes les informations sur la taille et le type d'un fichier.

REMARQUES :

— L'adresse retournée est un entier compris entre –32768 et 32767. Si elle est négative, on obtient l'adresse absolue en ajoutant 65536.

— /Variable/ doit être définie avant l'appel de VARPTR, sinon une erreur est provoquée.

— Si /Variable/ correspond au premier élément d'un tableau, tous ses éléments doivent être assignés avant l'appel, car l'emplacement mémoire d'un tableau se modifie au fur et à mesure de son remplissage.

— Pour pouvoir utiliser l'adresse retournée par VARPTR, il faut connaître la méthode de stockage des variables en mémoire. En règle générale, VARPTR pointe directement sur le contenu de la variable, défini comme suit :

Une variable entière est stockée sur deux octets consécutifs, octet de poids faible en premier.

Une variable en simple précision suit la même règle, mais sur quatre octets.

Une variable en double précision est stockée sur huit octets se suivant en mémoire.

Dans un tableau numérique, chaque élément occupe la place correspondant au type du tableau, et tous les éléments sont stockés du bas de la mémoire vers le haut en partant de l'indice le plus faible : Dans un tableau de variables en simple précision A!( ), VARPTR renvoie l'adresse du premier des quatre octets servant à définir A!(0). Pour passer à l'indice suivant, on ajoute 4 à l'adresse retournée.

Le cas des variables chaînes est plus complexe : L'adresse retournée par VARPTR correspond à une suite de trois octets appelée "descripteur de chaîne". Le premier contient la longueur de la chaîne, et les deux suivants l'adresse où la chaîne est effectivement stockée en mémoire, octet de poids faible en premier (voir PEEK).

Enfin, dans le cas d'un tableau de chaînes, VARPTR renvoie l'adresse de ce descripteur pour le premier élément du tableau.

***Exemple*** : Lecture de l'adresse d'une chaîne :

```
10 INPUT"VOTRE NOM ";A$
20 POINTEUR=VARPTR(A$)+1
30 ADRESSE=PEEK(POINTEUR)+(PEEK(POINTEUR+1)*256)
40 ?:?"ADRESSE DE LA CHAINE :";ADRESSE
50 END
```

### 3.4.2. Fonctions mathématiques

Pour toutes les fonctions mathématiques du MSX, le calcul se fait en double précision, quelque soit le type de l'argument employé.

#### ABS

*Syntaxe* : ABS(A)

*Application* : Renvoie la valeur absolue de A.

*Voir aussi* : SGN

***Exemple*** : ABS(10) renvoie 10, ABS(–10) également :

#### ATN

*Syntaxe* : ATN(A)

*Application* : ATN calcule l'angle en radians dont A est la tangente. Le résultat est compris entre –PI/2 et PI/2.

REMARQUE : ATN est l'inverse de TAN.

*Voir aussi* : SIN, COS, TAN.

#### COS

*Syntaxe* : COS(A)

*Application* : Renvoie le cosinus de l'angle A, qui doit être exprimé en radians.

*Voir aussi* : SIN, ATN, TAN.

**EXP**

*Syntaxe*: EXP(A)

*Application*: Calcule l'exponentielle de A en base e, c'est-à-dire e à la puissance A (e^A). A doit être inférieur à 146.

REMARQUE: EXP estla fonction inverse de LOG.

*Voir aussi*: LOG

**LOG**

*Syntaxe*: LOG(A)

*Application*: Retourne le logarithme népérien du nombre A.

REMARQUE: A doit être supérieur à 0.

*Voir aussi*: EXP.

**SGN**

*Syntaxe*: SGN(A)

*Application*: Retourne un nombre en fonction du signe de A:

— Si A<0, SGN(A) est égal à −1.

— Si A>0, SGN(A) est égal à 1.

— Si A=0, SGN(A)=0

*Voir aussi*: ABS

**SIN**

*Syntaxe*: Calcule le sinus de l'angle A, celui-ci étant exprimé en radians.

*Voir aussi*: COS, TAN, ATN.

**SQR**

*Syntaxe*: SQR(A)

*Application*: SQR calcule la racine carrée de A, avec A supérieur ou égal à 0.

REMARQUE: Cette fonction est l'inverse de A^2.

**TAN**

*Syntaxe*: TAN(A)

*Application*: Calcule la tangente de A, celui-ci étant exprimé en radians.

*Voir aussi*: ATN, SIN, COS.

### 3.4.3. Le traitement des chaînes de caractères

## ASC

*Syntaxe* : ASC(A$)

*Application* : Comme il est dit au paragraphe 1.3.1. consacré au clavier, chaque caractère correspond à un code compris entre 32 et 255 appelé code ASCII. La fonction ASC renvoie le code ASCII du premier caractère de la chaîne A$. Si A$ est une chaîne vide, une erreur est provoquée.

REMARQUES :

— Le jeu de caractères du MSX est répertorié à l'appendice 1.

— ASC est la fonction inverse du CHR$.

*Voir aussi* : CHR$

***Exemple*** : Affichage du code d'un caractère :

```
10 CLS
20 ? "ENTRER UN CARACTERE "
30 X$=INPUT$(1)
40 ?:?"CARACTERE: ";X$;"  CODE ASCII: ";ASC(X$)
50 ?
60 GOTO 20
```

(ASCII opposé de CHR$)

## CHR$

*Syntaxe* : CHR$(A)

*Application* : CHR$ renvoie le caractère correspondant au code ASCII A. L'argument peut être un entier ou une expression numérique contenant des variables préalablement définies.

REMARQUES :

— Les caractères du jeu ASCII ont un code compris entre 32 et 255, et peuvent être affichés directement par CHR$.

— On a vu dans le paragraphe 1.4.3. que les codes dont la valeur est inférieure à 32 correspondent à des fonctions spéciales du Basic, accessibles par le clavier à l'aide de la touche CONTROL. La fonction CHR$ permet d'utiliser ces codes de contrôle dans un programme, en les imprimant comme des caractères :
PRINT CHR$(7) ; par exemple correspond à CTRL/G.

*Voir aussi* : ASC.

***Exemple*** : Affichage du jeu de caractères :

```
40 CLS
50 PRINT"JEU DE CARACTERES:":PRINT
60 FOR I%=33 TO 255
70 PRINT CHR$(I%);" ";
80 NEXT
```

Démonstration des codes de contrôle.

```
10 CLS
15 FOR I=1 TO 9:?CHR$(31);:NEXT
20 ?"BONJOUR";
30 ?CHR$(31);
40 ?"JE SUIS";
50 ?CHR$(31);
60 ?"L'EDITEUR";
70 ?CHR$(30);
80 ?"DU MSX";
90 ?CHR$(30);
100 ?"BASIC";
110 X$=INPUT$(1)
120 ?CHR$(11);
130 ?"TERMINE"
```

Les codes 30 et 31 correspondent aux commandes de déplacement du curseur vers le haut et vers le bas, ligne par ligne (voir § 1.4.3). On les utilise ici pour positionner du texte sur l'écran.

## INSTR

*Syntaxe* : INSTR ([N,],A$,X$)

*Application* : Recherche la chaîne X$ à l'intérieur de A$, à partir de la position N (N compris entre 0 et 255). La valeur retournée est 0 si X$ n'est pas trouvée, ou la position du premier caractère de X$ dans A$.

REMARQUES :

— A$ et X$ peuvent être des variables ou des constantes de chaîne.

— Si N est omis, la recherche démarre à partir du premier caractère de A$.

— Si N est supérieur à la longueur de A$, INSTR renvoie 0.

— Si X$ est une chaîne vide, INSTR renvoie N ou 1 si N est omis.

***Exemple*** : Recherche d'un caractère dans une chaîne :

```
10 CLS
20 INPUT "ENTREZ VOTRE NOM ";NOM$
30 ?:?"ENTREZ LE CARACTERE CHERCHE ";
40 X$=INPUT$(1):?X$
50 A=INSTR(NOM$,X$)
55 IF A=0 THEN 80
60 ?:?"LE CARACTERE ";X$;" SE TROUVE EN ";A;"DANS ";NOM$
70 GOTO 30
80 ?:?"LE CARACTERE ";X$;" N'EST PAS DANS ";NOM$
90 GOTO 30
```

## LEFT$

*Syntaxe*: LEFT$(A$,N)

*Application*: LEFT$ extrait un sous-chaîne de A$ constituée des N premiers caractères à gauche de A$. N doit être un entier positif inférieur à 255 ou une expression numérique équivalente.

REMARQUES:

— Si N=0, LEFT$ renvoie une chaîne vide.

— Si N est supérieur à la longueur de A$, la totalité de A$ est renvoyée par la fonction.

*Voir aussi*: RIGHT$, MID$.

***Exemple***:

```
10 CLS
20 INPUT"DONNEZ VOTRE NOM ";A$
30 FOR I=1 TO LEN(A$)
40 PRINT LEFT$(A$,I)
50 NEXT I
```

Longueur de A$
affiche de gauche à droite.
voir mid$

## LEN

*Syntaxe*: LEN(A$)

*Application*: Calcule la longueur de la chaîne A$, en nombre de caractères. Si A$ est une chaîne vide (" "), alors LEN(A$)=0.

REMARQUE: Tous les caractères, y compris les codes de contrôle non imprimables et les espaces sont pris en compte par LEN.

***Exemple***:

```
 5 CLS
10 ?:INPUT "DONNEZ UN MOT DE CINQ LETTRES";A$
20 IF LEN(A$)=5 THEN GOTO 50
30 ?:?"ERREUR, CINQ LETTRES SVP"
40 GOTO 10
50 ?:?A$
60 GOTO 10
```

### MID$

*Syntaxe*: MID$(A$,N[,M])

*Application*: MID$ est la plus puissante des fonctions de traitement de chaîne du MSX-BASIC. Elle permet l'extraction de n'importe quelle partie d'une chaîne de caractères: La chaîne retournée contiendra M caractères de A$, en commençant par le Nième.

REMARQUES:

— N et M doivent êtr des entiers compris entre 1 et 255.

— Si M est omis ou si M est supérieur au nombre de caractères compris entre le Nième et la fin de A$, MID$ renvoie la totalité de ceux-ci.

— Si N est supérieur à la longueur de A$, MID$ renvoie une chaîne vide.

*Voir aussi*: LEFT$, RIGHT$.

***Exemples***:

```
10 A$="123456789"
20 ? MID$(A$,2,5)        nous imprimera 23456
```

Voici, pour démontrer les possibilités de MID$, un petit programme qui utilise cette fonction de manière amusante.

```
10 REM DEFILEMENT TEXTE
20 CLS
30 A$="ECRAN UTILISANT LES POSSIBILITES DE MID$ CECI EST UN DEFI
LEMENT DE TEXTE A L'ECRAN UTILISANT LES POSSIBILITES DE MID$ "
40 P=41
50 LOCATE 0,0
60 PRINT MID$(A$,P,40)
70 P=P+1
80 IF P=LEN(A$)-41 THEN P=1
90 GOTO 50
```

**Explications:**

— 30: Création d'une chaîne A$ contenant le texte à imprimer.

— 50: Positionnement du curseur en haut et à gauche de l'écran (voir LOCATE § 4.1.1).

— 60: Impression de 40 caractères de A$, à partir du Pième.

— 70: Incrément de P.

— 80: Lorsque toute la chaîne a été lue, on réinitialise P à 1.

En modifiant A$ et la ligne 50, ce programme pourra être utilisé par le lecteur pour d'autres applications.

## RIGHT$

*Syntaxe*: RIGHT$(A$,N)

*Application*: RIGHT$ extrait une sous-chaîne de A$ constituée des N derniers caractères à droite de A$. N doit être un entier positif ou une expression numérique équivalente.

REMARQUES:

— Si N=0, LEFT$ renvoie une chaîne vide.

— Si N est supérieur à la longueur de A$, la totalité de A$ est renvoyée par la fonction.

*Voir aussi*: LEFT$, MID$.

## SPACE$

*Syntaxe*: SPACE$(N)

*Application*: SPACE$ retourne une chaîne composée de N espaces.

REMARQUE: N doit être compris entre 0 et 255.

*Voir aussi*: STRING$.

***Exemple***:

```
PRINT "MSX"SPACE$(20)"BASIC
```

## STR$

*Syntaxe*: STR$(N)

*Application*: Convertit une constante ou une expression numérique en chaîne de caractères.

REMARQUES:

— Tout nombre positif est précédé d'un espace qui apparaît à l'impression (il s'agit de l'emplacement du signe). Lors de sa conversion en chaîne, cet espace est conservé.

— Cette fonction s'utilise pour effectuer sur des nombre des traitements réservés aux chaînes. VAL permet l'opération inverse.

*Voir aussi*: VAL

***Exemple***:

```
10 A=8.584:B=3.999
20 A$=STR$(A):B$=STR$(B)
30 A$=LEFT$(A$,4):B$=LEFT$(B$,4)
40 ?"A et B sur une seule décimale ";A$;" ";B$
```

**Explication :**

— 20 : A et B sont convertis en A$ et B$.

— 30 : Prise en compte des quatre premiers caractères de A$ et B$ (un espace et trois chiffres).

— 40 : Affichage de A$ et B$. Pour les convertir à nouveau en valeurs numériques, voir VAL.

## STRING$

*Syntaxes* :

```
STRING$(N,X$)
STRING$(N,X)
```

*Application* : Renvoie une chaîne contenant N fois le premier caractère de X$, ou dans sa deuxième syntaxe le caractère de code ASCII X.

REMARQUE : STRING$ est très pratique pour souligner du texte ou dessiner rapidement des cadres de présentation à l'écran.

*Voir aussi* : SPACE$.

***Exemple*** :

```
10 CLS
20 A$="-"
30 ? STRING$(10,A$)"RESULTATS"STRING$(10,A$)
40 ? STRING$(29,A$)

Donnera:

----------RESULTATS----------
-----------------------------
```

## VAL

*Syntaxe* : VAL(A$)

*Application* : Retourne l'équivalent numérique d'une chaîne de caractères :

— Si A$ contient un nombre, VAL renvoie ce nombre.

— Si A$ contient un nombre et des caractères, VAL ne renverra le nombre qui s'il est placé au début de A$.

— Si le contenu de A$ ne peut être évalué, VAL renvoie 0.

REMARQUES :

— VAL ne tient pas compte des espaces dans A$.

— Les préfixes de conversion &B, &O et &H sont interprétés par la fonction VAL.

*Voir aussi*: STR$.

***Exemples***:

```
VAL("5 FRANCS")=5

VAL("X=3*4")=0

VAL("6/12")=6

VAL("&B11101111")=255
```

# 4. La gestion des périphériques

Avant d'aborder cette section, il est indispensable de clarifier la notion de "Périphériques", prise ici au sens le plus large :

Toute connexion effectuée entre la "console", terme que nous emploierons pour désigner l'unité centrale de l'ordinateur et un élément extérieur à celle-ci peut être considérée comme une entrée ou une sortie vers un périphérique. Ainsi, lorsqu'on tape un caractère au clavier, lequel s'affiche ensuite à l'écran, on effectue une entrée du périphérique "clavier" vers la console qui à son tour effectue une sortie vers le périphérique "écran".

Bien entendu, il est évident que l'imprimante, un mini-cassette ou d'éventuelles disquettes sont des périphériques, mais il en va de même pour le clavier, l'écran, ou le port d'expansion.

Toutes les machines respectant le standard MSX sont dotées d'un nombre très important de périphériques en standard:

— Un ou deux lecteurs de cartouches.
— Deux entrées pour manettes de jeux.
— Une sortie parallèle imprimante.
— Une sortie pour mini-cassette.
— Une sortie audio.
— Une sortie vidéo composite.
— Une sortie pour écran couleur RVB ou péritel.
— Un bus d'extension pour tous les types de raccordements.

## 4.1. L'ÉCRAN

Cette section est consacrée à l'écran en mode texte, ainsi qu'à toutes les commandes et fonctions correspondantes. L'écran haute-résolution graphique est traité en détail au chapitre 5.

### 4.1.1. Instructions

#### CLS

*Syntaxe*: CLS

*Application*: Efface l'écran, et place le curseur en haut et à gauche.

REMARQUE: CLS efface tout l'écran, quel que soit son contenu, graphique ou texte.

#### COLOR

*Syntaxe*: COLOR [/Avant-plan/][,/Fond/][,/Bordure/]

*Application*: COLOR sert à définir la couleur utilisée pour l'affichage. Les trois paramètres permettent de spécifier:

— La couleur du texte avec /Avant-plan/.
— La couleur du fond avec /Fond/.
— La couleur du bord de l'écran avec /Bordure/.

Le MSX-BASIC gère l'affichage dans 16 couleurs différentes :

0 Transparent
1 Noir
2 Vert moyen
3 Vert clair
4 Bleu foncé
5 Bleu clair
6 Rouge foncé
7 Cyan
8 Rouge moyen
9 Rouge clair
10 Jaune foncé
11 Jaune clair
12 Vert foncé
13 Magenta
14 Gris
15 Blanc

REMARQUES :

— En mode texte, la sélection d'une couleur pour l'écriture ou pour le fond modifie instantanément la couleur de tout l'écran, qui prend les valeurs choisies.

— Les règles d'apparition des couleurs sont différentes selon que l'on est en mode texte ou graphique, et suivant les paramètres choisis dans SCREEN (voir cette instruction et le chapitre 5).

— A la mise sous tension, COLOR 15,4,7 est exécutée par défaut.

— Il est possible d'omettre deux des paramètres de l'instruction COLOR. Dans ce cas, les valeurs courantes sont conservées.

*Voir aussi* : SCREEN

***Exemple*** : COLOR 15,,14 sélectionne un affichage en Blanc avec une bordure grise. Le fond existant est conservé.

## LOCATE

*Syntaxe* : LOCATE [X] [,Y] [,/Curseur/]

*Application* : Fixe la position du curseur et donc la position d'écriture à l'écran en mode texte. X représente le numéro de colonne et doit être compris entre 0 et 39. Y représente le numéro de ligne et doit être compris entre 0 et 23.

REMARQUES :

— /Curseur/ valide ou non l'affichage du curseur clignotant.
Si /Curseur/ est égal à 1, le curseur est visible.
Si /Curseur/ est égal à 0, le curseur disparaît.
Si /Curseur/ est omis, le curseur reste dans le mode en vigueur.

— Si Y est omis, LOCATE ne modifie que la position à l'intérieur de la ligne.

— Si X est omis, seul le numéro de ligne est modifié.

— Si X ou Y sont des valeurs hors limites, une erreur est provoquée.

— Le coin supérieur gauche de l'écran a pour coordonnées 0,0.

*Voir aussi*: PRINT, PRINT USING

***Exemples***:

| | |
|---|---|
| `LOCATE 10,15` | place le curseur à la 16e ligne, 11e colonne. |
| `LOCATE 10` | place le curseur sur la 11e colonne, sans changement de ligne. |
| `LOCATE ,10` | place le curseur sur la 11e ligne, sur la même colonne. |
| `LOCATE 1,1,0` | place la position d'écriture dans le coin supérieur gauche de l'écran, curseur invisible. |
| `LOCATE 1,,0` | place la position d'écriture sur la première colonne, sans changement de ligne, curseur invisible. |

L'instruction LOCATE s'utilise principalement avec PRINT:

```
10 REM DEPLACEMENT D'UN MOBILE A L'ECRAN
20 CLS:X=20:Y=10:A$="[*]"
25 LOCATE X,Y:PRINT A$
30 XM=X:YM=Y
40 X$=INKEY$
50 IF X$="" THEN 40
60 A=ASC(X$)
70 IF A<28 OR A>31 THEN 40
80 LOCATE XM,YM:PRINT "   "
90  IF A=28 AND X<>39 THEN X=X+1
100 IF A=29 AND X<>0  THEN X=X-1
110 IF A=30 AND Y<>0  THEN Y=Y-1
120 IF A=31 AND Y<>23 THEN Y=Y+1
130 LOCATE X,Y:PRINT A$
140 GOTO 30
```

**Explications :**

— 20: Effacement de l'écran, initialisation des variables.

— 25: Positionne le curseur au centre de l'écran, affiche la chaîne A$.

— 30: XM et YM mémorisent X et Y, les coordonnées où A$ est affichée.

— 40: Saisie d'un caractère au clavier.

— 50: Si aucune touche n'est actionnée, on recommence la saisie.

— 60: On mémorise dans A le code ASCII du caractère saisi.

— 70: Si ce code ne correspond pas à l'un des quatre codes de contrôle envoyés par les touches de déplacement du curseur, on reprend la saisie (ces codes sont répertoriés au paragraphe 1.4.3).

— 80 : On replace le curseur au dernier emplacement occupé par la chaîne A$, qu'on efface en la remplaçant par des espaces.

— 90-120 : Suivant le sens de déplacement choisi avec les flèches du clavier (droite, gauche, haut ou bas), on modifie la valeur des coordonnées X et Y en interdisant un dépassement des valeurs limites correspondant aux bords de l'écran.

— 130 : Positionne le curseur au nouvel emplacement défini par X et Y, puis affiche le mobile A$.

— 140 : Retour à la mémorisation des coordonnées de la ligne 30.

## SCREEN

*Syntaxe* :

```
SCREEN [/Mode/][,/Taille des sprites/][,/Sonorisation clavier/]
       [,/Vitesse K7/][,/Type d'imprimante/][,ENTRELACEMENT/]
```

*Application* : SCREEN est une instruction servant à initialiser l'écran, le graphique et certains périphériques. Tous les paramètres peuvent être omis en les remplaçant par une virgule : Dans tous les cas la valeur courante est prise par défaut.

— /Mode/ doit être un entier compris entre 0 et 4. On l'utilise pour sélectionner un des quatre types d'affichages disponibles en MSX-BASIC. (MSX2 + 5 à 8)

0. : Mode texte, 40 colonnes sur 24 lignes. C'est le mode par défaut à la mise sous tension de l'ordinateur.

1 : Mode texte, 32 colonnes sur 24 lignes. Les caractères sont légèrement élargis dans ce mode.

2 : Mode graphique haute-résolution, 256 × 192 points. Il est impossible de passer sur l'écran graphique en mode direct : Le MSX-BASIC retourne instantanément au mode texte en sortie de programme (voir le graphique, chapitre 5).

3 : Mode graphique basse-résolution, 64 × 48 blocs. Le fonctionnement est identique à celui de la haute résolution.

Dans tous les cas, la modification du mode en vigueur dans un programme implique l'effacement de l'écran. Il est toutefois possible d'afficher du texte sur un écran haute-résolution (voir § 5.2). L'inverse n'est pas vrai, et les instructions du graphique provoquent des erreurs en mode texte.

Le fonctionnement de la couleur suit des règles particulières dans les deux modes graphiques, voir chapitre 5.

— /Taille des sprites/ permet de sélectionner quatre tailles différentes pour les "sprites" (voir utilisation détaillée des sprites au § 5.3) :

0: 8 × 8 cases, taille standard: C'est le mode par défaut.
1: 8 × 8 cases, mode double taille.
2: 16 × 16 cases, taille standard.
3: 16 × 16 cases, mode double taille.

Chaque fois qu'un nouveau choix pour la taille des sprites est effectué par SCREEN, la variable SPRITE$ qui mémorise toutes les définitions de formes est réinitialisée.

— Chaque fois qu'une touche est pressée au clavier, un "clic" est envoyé sur la sortie son de l'ordinateur, ou sur le haut-parleur interne s'il en est pourvu. /Sonorisation clavier/ permet de supprimer ce son si elle est égale à 0. Toute autre valeur valide la sonorisation.

— /Vitesse K7/ fixe la vitesse de transmission des données lors des sauvegardes sur un magnétophone à cassettes. Deux valeurs sont disponibles:

1: La vitesse est de 1200 Bauds, ce qui assure une fiabilité maximum (un Baud=1 Bit/seconde). C'est la valeur par défaut à la mise sous tension.

2: La vitesse est de 2400 Bauds. Il convient d'utiliser un support magnétique de bonne qualité si cette option est choisie.

Il est également possible de définir une vitesse de transmission par l'intermédiaire d'une option de la commande CSAVE (voir § 4.3).

*Note*: Quelque soit la vitesse de sauvegarde choisie, le MSX-BASIC la reconnaît automatiquement à la relecture.

— Si l'utilisateur dispose d'une imprimante spéciale pouvant afficher les symboles graphiques du jeu de caractères MSX, /Type d'imprimante/ doit valoir 0. Dans tous les autres cas, les symboles graphiques sont remplacés par des espaces avant l'impression.

- 1: ON  0: OFF très pratique en 60 Hz,

***Exemples***:

| | |
|---|---|
| `SCREEN 1,,0` | Sélectionne le mode texte en 32 colonnes, et supprime le clic clavier. |
| `SCREEN 0` | Réinitialise en mode texte 40 colonnes. |
| `SCREEN ,,,2` | Modifie la vitesse de transmission vers le K7. |

**VDP**

*Syntaxe*: VDP(N)

*Application*: Permet l'accès direct au circuit contrôlant les paramètres de base de l'affichage vidéo. N doit être compris entre 0 et 7, et donne

accès à 8 registres différents ne fonctionnant qu'en écriture. Si N=8, VDP renvoie un octet contenant le statut du processeur.

VDP(10)=Ø 60Hz / VDP(10)=2 50Hz

## VPOKE

*Syntaxe*: VPOKE /Adresse/, /Octet/

*Application*: VPOKE est totalement identique à l'instruction POKE, mais sert à adresser les 16 Kilo-octets de la mémoire écran. /Octet/ doit être compris entre 0 et 255, et /Adresse/ entre 0 et 16383.

REMARQUES:

— La mémoire vidéo du MSX fonctionne en haute-résolution : VPOKE affiche des octets, pas des caractères.

— VPOKE permet les accès directs à la mémoire écran sans nuire à la compatibilité des programmes entre différentes machines répondant au standard.

*Voir aussi*: VPEEK.

## WIDTH

*Syntaxe*: WIDTH /Nombre de colonnes/

*Application*: Permet de modifier la largeur de l'écran en mode texte. /Nombre de colonnes/ doit être compris entre 1 et 40 en mode 40 × 24 ou entre 1 et 32 en mode 32 × 24 (voir SCREEN).

REMARQUE: Une fois la largeur de l'écran choisie, toutes les instructions d'affichage ne fonctionnent qu'à l'intérieur de la fenêtre définie avec WIDTH.

***Exemple***:

```
10 CLS:WIDTH 20
20 FOR I=1 TO 400
30 ?"A";
40 NEXT
50 X$=INPUT$(1)
```

### 4.1.2. Fonctions

## CSRLIN

*Syntaxe*: CSRLIN

*Application*: Retourne la position verticale du curseur, comprise entre 0 et 23.

*Voir aussi*: LOCATE, POS.

**POS**

*Syntaxe*: POS(I)

*Application*: Retourne la position horizontale du curseur, comprise entre 0 et 39 en mode texte 40 colonnes.

REMARQUE: I peut prendre n'importe quelle valeur.

*Voir aussi*: LOCATE, CSRLIN.

***Exemple***:

```
LOCATE 1,8:?"LIGNE:";CSRLIN+1;" COLONNE:";POS(0)+1
```

## 4.2. LE CLAVIER

Cinq touches donnant accès à dix fonctions programmables sont disponibles au clavier du MSX (voir § 1.3.1). Chacune d'entre elles est associée à une chaîne de caractères correspondant à une instruction du Basic MSX. On peut par l'intermédiaire de l'instruction KEY modifier leur affectation, et l'afficher sur la 24ᵉ ligne de l'écran. Il est également possible de saisir directement une fonction par son numéro en utilisant un déroutement par la déclaration ON KEY.

Lors de l'écriture des programmes, il est pratique de choisir pour les dix touches de fonctions les dix instructions les plus couramment utilisées, pour accélérer l'entrée du texte.

L'emploi de ces touches à l'intérieur d'un programme facilite la gestion du clavier et permet la création de menus personnalisés: Il est en effet plus simple d'associer des fonctions à un pavé de cinq touches toujours identique en haut et à gauche du groupe alphanumérique, plutôt qu'à des caractères dispersés sur la partie centrale du clavier.

### 4.2.1. Programmation des touches de fonction

**KEY**

*Syntaxe*: KEY /N° de la touche/, /Chaîne de caractères/

*Application*: Affecte la chaîne de caractères à la touche dont le numéro est spécifié. /N° de la touche/ doit être compris entre 1 et 10. La longueur de la chaîne ne doit pas excéder 15 caractères.

REMARQUES :

— /Chaîne de caractères/ peut être une variable, une constante, ou une expression :

```
A$="MSX-BASIC":KEY 1,A$

KEY 1,"MSX-BASIC"

KEY 1,"MSX"+"-BASIC"
```

sont des signes de commandes équivalentes.

— Si la chaîne correspond à une instruction du Basic, une pression sur la touche ne fait qu'afficher la commande à la position du curseur, sans l'exécuter :

`KEY 1,"PRINT"` Assigne la chaîne PRINT à la touche N° 1.

— Si la chaîne se termine par le code de contrôle correspondant à la touche RETURN (code 13, voir § 1.4.3), la commande s'exécute automatiquement :

`KEY 1,"LIST"+CHR$(13)` La touche 1 liste le programme en mémoire.

— Les fonctions de 6 à 10 sont obtenues en combinant l'action de SHIFT avec les cinq touches spécialisées.

## KEY LIST

*Syntaxe* : KEY LIST

*Application* : Liste à l'écran par ordre numérique croissant les chaînes de caractères associées aux dix touches de fonction programmables.

REMARQUE : Les codes de contrôle ne sont pas visualisés par l'instruction.

## KEY ON/OFF

*Syntaxe* : KEY ON
KEY OFF

*Application* : A la mise sous tension de l'ordinateur, la 24e ligne de l'écran est automatiquement réservée pour l'affichage des chaînes associées aux touches de fonction programmable.

KEY OFF supprime cet affichage et libère la ligne.

KEY ON rétablit l'affichage des fonctions.

REMARQUE : Cinq fonctions à la fois sont affichées à l'écran, suivant que la touche SHIFT est appuyée ou non.

### 4.2.2. Saisie des touches de fonction

Comme pour ON INTERVAL et ON STOP, le MSX-BASIC gère un branchement automatique prioritaire sur les 10 touches de fonctions programmables (voir principe au § 3.3.4).

#### ON KEY GOSUB

*Syntaxe*: ON KEY GOSUB [/N° 1] [,/N° 2]... [,/N° 10/]

*Application*: Il est possible de dérouter l'action des dix touches de fonction pour effectuer un branchement vers un sous-programme avec la déclaration ON KEY GOSUB. La liste de numéros donnée en argument permet le branchement vers un sous-programme par fonction: Le premier numéro de la liste correspond à la touche numéro 1, le deuxième à la touche numéro 2, etc... On peut sauter une touche en remplaçant un des numéros de ligne par une virgule.

ON KEY GOSUB 100,,200,,300 déclare un déroutement pour les touches de fonction 1, 3 et 5.

Comme pour les autres types de déroutements, l'instruction KEY(N) ON valide cette déclaration (voir KEY(N) ON/OFF/STOP).

REMARQUES:

— Chaque fois qu'un déroutement sur une touche est effectué, un KEY(N) STOP est automatiquement exécuté. En retour de sous-programme, un KEY(N) ON supprime l'inhibition des déroutements.

— Les déroutements ne fonctionnent qu'en mode programme.

*Voir aussi*: KEY(N) ON/OFF/STOP

***Exemple***: Création d'un menu.

```
10 CLS:KEY OFF
20 ON KEY GOSUB 200,300,400,500,,,,,,600
30 KEY(1) ON
40 KEY(2) ON
50 KEY(3) ON
60 KEY(4) ON:KEY(10)ON
70 LOCATE 14,1:PRINT"MENU":?:?
80  ?"F1  OPTION NUMERO 1"
90  ?"F2  OPTION NUMERO 2"
100 ?"F3  OPTION NUMERO 3"
110 ?"F4  OPTION NUMERO 4"
120 ?"F10 OPTION NUMERO 5"
130 LOCATE 1,20:?"TAPEZ UNE TOUCHE DE FONCTION"
140 GOTO 130
150 END
200 REM OPTION 1
210 ?:?"SOUS-PROGRAMME No 1"
```

```
220 GOTO 1000
300 REM OPTION 2
310 ?:?"SOUS-PROGRAMME No 2"
320 GOTO 1000
400 REM OPTION 3
410 ?:?"SOUS-PROGRAMME No 3"
420 GOTO 1000
500 REM OPTION 4
510 ?:?"SOUS-PROGRAMM_ No 4"
520 GOTO 1000
600 REM OPTION 5
610 ?:?"SOUS-PROGRAMME No 5"
620 GOTO 1000
1000 REM RETOUR MENU
1010 RETURN
```

**Explications :**

Le programme affiche à l'écran les options disponibles, liées aux cinq touches de fonction. La frappe d'une touche provoque le déroutement vers un des sous-programmes, qui affiche un message en fonction du numéro de la touche, puis revient dans le menu.

— 10 : KEY OFF supprime l'affichage des touches sur la 24e ligne.

— 20 : Déclaration de déroutement pour les touches 1, 2, 3, 4 et 10.

— 30-60 : Valide la déclaration.

— 70-120 : Affichage du menu.

— 130-140 : Boucle infinie ne servant qu'à rester dans le programme principal.

— 200-260 : Sous-programmes appelés en cas de déroutement.

— 1000 : Retour dans la boucle infinie après déroutement.

## KEY(N) ON/OFF/STOP

*Syntaxe* : KEY(N) ON
KEY(N) OFF
KEY(N) STOP

*Applications* :

— KEY(N) ON valide pour la touche de fonction N le déroutement par la déclaration ON KEY GOSUB.

— KEY(N) OFF supprime la prise en compte de ON KEY GOSUB pour la touche N.

— KEY(N) STOP mémorise l'action sur la touche spécifiée mais ne valide le déroutement qu'après rencontre de KEY(N) ON.

REMARQUE : N doit être compris entre 1 et 10.

## 4.3. LE MAGNÉTOPHONE A CASSETTES

Des instructions du MSX-BASIC sont spécialisées dans la gestion de l'interface cassette : Il est possible de lire et charger des zones mémoire, des programmes, ou des fichiers séquentiels. Un chapitre est consacré exclusivement à ces derniers, plus complexes à mettre en œuvre (voir § 4.5).

*Note* : Certaines des commandes de sauvegarde fonctionnent également avec des disquettes lorsque le Basic disque est installé (voir MSX-DOS § 4.6). Lorsque le cas se présente, les remarques en font état.

### MOTOR ON/OFF

*Syntaxe* : MOTOR
MOTOR ON
MOTOR OFF

*Application* : Si le magnétocassette est équipé d'une télécommande (prise "remote"), l'instruction MOTOR sans argument stoppe ou démarre le défilement de la bande magnétique. En spécifiant ON ou OFF, on peut imposer le mode sans tenir compte de l'état précédent.

### BSAVE

*Syntaxe* :

BSAVE "/Périphérique/[/Nom fichier/]",/Adresse début/,
/Adresse fin/[,/Adresse démarrage/]

*Application* : BSAVE permet la sauvegarde d'une zone mémoire sur cassette. Cette zone mémoire peut contenir un programme en langage machine ou simplement des données numériques (sauvegarde de la zone des variables, etc...).

— /Périphérique/ représente la sortie vers laquelle sont envoyées les données (voir OPEN, § 4.5). Dans le cas de BSAVE, seul CAS : est autorisé.

— /Nom fichier/ est optionnel, il permet de donner un nom à la zone sauvegardée.

— /Adresse début/ et /Adresse fin/ servent à sélectionner la longueur de la zone mémoire à sauvegarder.

— /Adresse démarrage/ permet de spécifier une adresse à laquelle l'exécution doit démarrer après chargement de la zone par BLOAD : Cette option n'est utile que lors de la sauvegarde de programmes exécutables. Si /Adresse démarrage/ est omise, /Adresse début/ est prise par défaut.

REMARQUES :

— Les adresses peuvent être entrées directement en hexadécimal, en les faisant précéder de &H.

— Si le Basic disque ou le MSX-DOS est implanté sur la machine, BSAVE permet la sauvegarde sur disque en donnant comme périphérique le nom des lecteurs de disquettes :

A : Pour le lecteur principal.

B : Pour le suivant.

*Voir aussi* : BLOAD.

***Exemples*** :

— BSAVE "CAS:TEST",49500,65535,51000 sauvegarde la zone mémoire comprise entre les adresses 49500 et 65535 sous le nom TEST. Lors du chargement par BLOAD, l'option R force l'exécution à l'adresse 51000.

— BSAVE "CAS:",&HA000,&HB000 sauvegarde sans la nommer une zone sur K7.

## BLOAD

*Syntaxe* : BLOAD "/Périphérique/[/Nom fichier/]" [,R] [,/Offset/]

*Application* : BLOAD charge la zone ou le programme en langage-machine sauvegardé par BSAVE sur le cassette.

— /Périphérique/ doit être CAS:.

— /Nom fichier/ permet de sélectionner sur une bande la zone désirée : BLOAD ne charge que le fichier spécifié. En l'absence d'un nom, la première zone rencontrée est chargée.

— Enfin, l'option R force l'exécution d'un programme immédiatement après son chargement, à l'adresse fixée dans BSAVE.

REMARQUES :

— Lors du chargement, la zone mémoire se place aux mêmes adresses qu'avant la sauvegarde. L'option Offset permet de décaler la zone du nombre d'adresses spécifié.

— Comme pour BSAVE, BLOAD peut fonctionner avec des lecteurs de disquettes en remplaçant CAS: par A: ou B:.

*Voir aussi*: BSAVE.

**CSAVE**

*Syntaxe*: CSAVE "/Nom fichier/" [,/Vitesse de transmission/]

*Application*: Sauvegarde un programme basic sur le périphérique cassette. L'option /Vitesse de transmission/ permet de choisir entre deux vitesses d'écriture, comme dans SCREEN (voir § 4.1.1). La valeur 1 correspond à 1 200 Bauds, et 2 à 2 400 Bauds.

*Voir aussi*: CLOAD, CLOAD ?

**CLOAD**

*Syntaxe*: CLOAD "/Nom fichier/"

*Application*: Pour charger un programme Basic sauvegardé par CSAVE sur le cassette. Si /Nom fichier/ est omis, le premier programme rencontré sur la bande est chargé.

REMARQUES:

— CLOAD ferme tous les fichiers et vide la mémoire de l'ordinateur (comme NEW).

— La vitesse de transmission est sélectionnée automatiquement à la lecture par CLOAD.

— Lors de la recherche de programmes sur bande, CLOAD affiche SKIP suivi du nom pour tous les programmes ne correspondant pas à /Nom fichier/, et FOUND dans le cas contraire.

*Voir aussi*: CSAVE, CLOAD ?

**CLOAD ?**

*Syntaxe*: CLOAD ? ["/Nom fichier/"]

*Application*: Permet de vérifier le bon enregistrement par CSAVE d'un programme Basic sur cassette, en le comparant au contenu de la mémoire. En cas d'erreur, un message est affiché.

REMARQUE: Si /Nom fichier/ est omis, la comparaison s'effectue avec le premier programme rencontré sur le cassette.

*Voir aussi*: CSAVE, CLOAD.

## SAVE

*Syntaxe*: SAVE "/Périphérique/[/Nom fichier/]"

*Application*: SAVE sauvegarde un programme sur le périphérique spécifié en mode ASCII: Un programme Basic peut être décomposé en une suite d'octets dont le code ASCII est compris entre 0 et 255. Les instructions CSAVE et CLOAD sauvegardent les programmes sous une forme compressée, prenant moins de place sur la bande magnétique. Dans certains cas, il est indispensable d'effectuer une sauvegarde en mode ASCII, sans conversion en binaire: L'instruction MERGE, par exemple, ne fonctionne que sur des fichiers en ASCII. En l'absence de système d'exploitation de disquette, CAS: doit être utilisé pour sauvegarder sur cassette.

REMARQUES:

— /Nom fichier/ peut être omis.

— CAS: doit être remplacé par A: ou B: pour sauvegarder sur disquettes.

*Voir aussi*: LOAD, MERGE.

***Exemple***:

```
SAVE "CAS:TEST"
```

## LOAD

*Syntaxe*: LOAD "/Périphérique/[/Nom fichier/]"[,R]

*Application*: Charge un programme basic sauvegardé en mode ASCII par SAVE à partir du périphérique spécifié (CAS: pour le cassette).

REMARQUES:

— Si/Nom fichier/ est omis, le premier programme rencontré sur la cassette est chargé.

— LOAD ferme tous les fichiers et vide la mémoire.

— L'option R permet de lancer l'exécution automatiquement après chargement du programme.

— Comme pour BLOAD, il est possible de charger un programme à partir d'une disquette en remplaçant CAS: par A: ou B:.

*Voir aussi*: SAVE.

***Exemple***:

```
LOAD "CAS:TEST",R
```

## MERGE

*Syntaxe*: MERGE "/Périphérique/[/Nom fichier/]"

*Application*: Permet de fusionner le programme résidant en mémoire avec un autre se trouvant sur cassette. Les lignes de programme dont les numéros correspondent à celles du fichier en cours de chargement sont remplacées par ces dernières.

REMARQUES:

— /Périphérique/ doit être CAS: pour charger un programme à partir du cassette, A: ou B: pour les disquettes.

— /Nom fichier/ peut être omis: Dans ce cas, le premier programme lu sur la bande est fusionné au programme en mémoire.

— Pour qu'un programme puisse être chargé par la commande MERGE, il doit être en format ASCII sur la bande, et donc sauvegardé par SAVE.

*Voir aussi*: SAVE, LOAD.

***Exemple***:

```
MERGE "CAS:TEST"
```

## 4.4. LES MANETTES DE JEUX

Les deux interfaces pour manettes de jeux peuvent également être utilisées pour connecter une tablette graphique ou tout autre type de périphérique. Quatre fonctions et un déroutement permettent le traitement dans un programme Basic des informations renvoyées par ces périphériques:

```
ON STRIG GOSUB

STRIG(N) ON/OFF/STOP

STRIG(N)

STICK(N)

PDL(N)

PAD(N)
```

## ON STRIG GOSUB

*Syntaxe*: ON STRIG GOSUB [/N° 1/] [,/N° 2/]... [,/N° 5/]

*Application*: Il est possible de dérouter l'action sur le bouton poussoir d'une manette de jeu pour effectuer un branchement prioritaire vers un sous-programme avec la déclaration ON STRIG GOSUB. La liste de cinq numéros donnée en argument permet le branchement vers un sous-programme différent en fonction de l'interface utilisé : Le premier numéro de la liste correspond à la touche barre du clavier, qui remplace alors la manette, le deuxième et le quatrième numéro au bouton du Joystick N° 1, le troisième et le cinquième au Joystick N° 2. On peut sauter une touche en omettant un des numéros de ligne de la liste.

ON STRIG GOSUB ,,200,,300: déclare un déroutement pour les touches du Joystick N° 2.

ON STRIG GOSUB 100: saisit la touche barre.

Comme pour les autres types de déroutements, l'instruction STRIG(N) ON valide cette déclaration (voir STRIG(N) ON/OFF/STOP).

REMARQUES :

- Chaque fois qu'un déroutement sur une touche est effectué, un STRIG(N) STOP est automatiquement exécuté. En retour de sous-programme, un STRIG(N) ON supprime l'inhibition des déroutements.
- Les déroutements ne fonctionnent qu'en mode programme.

*Voir aussi*: STRIG(N) ON/OFF/STOP.

## STRIG(N) ON/OFF/STOP

*Syntaxe*: STRIG(N) ON
STRIG(N) OFF
STRIG(N) STOP

*Applications*:

- STRIG(N) ON valide pour la touche correspondant à N le déroutement par la déclaration ON STRIG GOSUB.
- STRIG(N) OFF supprime la prise en compte de ON STRIG GOSUB pour la touche N.
- STRIG(N) STOP mémorise l'action sur la touche spécifiée mais ne valide le déroutement qu'après rencontre de STRIG(N) ON.

REMARQUE : N doit être compris entre 0 et 4 :

N=0 correspond à la touche barre du clavier.

N=1 et N=3 au bouton poussoir du Joystick N° 1.

N=2 et N=4 à celui du Joystick N° 2.

*Voir aussi*: ON STRIG GOSUB.

***Exemple***: Saisie de la barre clavier.

```
10 ON STRIG GOSUB 100
20 STRIG(0) ON
25 CLS:A=0:B=1
30 LOCATE 1,1
40 PRINT "BOUCLE INFINIE! No DE TOUR:";A
50 A=A+1
60 GOTO 30
100 REM DEROUTEMENT
110 LOCATE 1,20
120 PRINT "APPUI SUR LA BARRE. No DE TOUR:";B
130 B=B+1
140 RETURN
```

La boucle infinie des lignes 30 et 60 ne peut être interrompue que par une action sur la touche barre: Le sous-programme de la ligne 100 incrémente alors un compteur B, indiquant un déroutement.

## STRIG(N)

*Syntaxe*: STRIG(N)

*Application*: Retourne le statut de la touche barre du clavier ou du bouton poussoir des manettes de jeu: Si la touche est appuyée, la fonction renvoie −1 et 0 dans le cas contraire.

REMARQUE: N doit être compris entre 0 et 4:

N=0 correspond à la touche barre du clavier.

N=1 et N=3 au bouton poussoir du Joystick N° 1.

N=2 et N=4 à celui du Joystick N° 2.

*Voir aussi*: ON STRIG GOSUB, STRIG(N) ON/OFF/STOP.

## STICK(N)

*Syntaxe*: STICK(N)

*Application*: Renvoie la direction du manche à balai sur la manette de jeu. N doit être compris entre 0 et 2: Si N=0, les touches de déplacement du curseur sont traitées comme un Joystick, si N=1 ou N=2, la manette de jeu connectée à l'interface 1 ou 2 est saisie. Les valeurs prises par la fonction permettent la prise en compte de 8 directions différentes:

0 Pas d'action
1 Haut
2 Haut droite
3 Droite
4 Bas droite
5 Bas
6 Bas gauche
7 Gauche
8 Haut gauche

REMARQUES :

— Pour obtenir les huit directions avec les touches de déplacement du curseur, il faut les appuyer simultanément deux à deux.

— STICK(N) s'utilise également pour saisir les déplacements d'un stylet sur une tablette graphique.

— En associant le déroutement sur la barre d'espacement du clavier et la saisie des touches de gestion du curseur, il est possible de traiter le clavier comme une manette de jeu. Ce principe est illustré dans le programme de dessin par les flèches du clavier servant d'exemple à l'instruction PSET (voir § 5.2).

*Voir aussi* : STRIG(N).

**PDL(N)**

*Syntaxe* : PDL(N)

*Application* : Renvoie une valeur en fonction de l'interface sur lequel un accessoire est connecté : Si N est impair, c'est le port Joystick N° 1 qui est employé, si N est pair le N° 2.

**PAD(N)**

*Syntaxe* : PAD(N)

*Application* : Retourne le statut d'un accessoire de type "paddle". N doit être compris entre 0 et 7. Les valeurs inférieures à 4 concernent l'interface Joystick N° 1, les quatre suivantes le N° 2.

— Si N=0 ou N=4, la fonction retourne −1 si l'utilisateur appuie sur une des touches du paddle, 0 dans le cas contraire.

— Si N=3 ou N=7, la fonction renvoie −1 ou 0 suivant que l'interrupteur principal du paddle soit pressé ou non.

— Si N=1 ou N=5, PAD(N) prend la valeur de la coordonnée horizontale où le paddle envoie le curseur.

— Si N=2 ou N=6, c'est la coordonnée verticale qui est retournée par la fonction.

REMARQUE : Les coordonnées verticales et horizontales ne sont renvoyées qu'après lecture de PAD(0) ou PAD(4), et ce uniquement dans le cas où le résultat est −1. Leurs limites correspondent à la résolution du graphique MSX, 256 × 192.

## 4.5. LE TRANSFERT DE DONNÉES

### 4.5.1. Généralités

Pour créer un fichier de données, il est indispensable de connaître une méthode permettant le stockage de variables sur un magnétophone à cassettes. De même, pour copier le contenu de l'écran sur une imprimante, il faut pouvoir adresser ces deux périphériques en lecture et en écriture de manière à transférer des données de l'un vers l'autre. Sur certains Basic, la connection entre l'unité centrale de l'ordinateur et les périphériques qui lui sont associés est fixe : Pour chaque type de transfert de données, une instruction spécialisée est prévue dans le langage. En MSX-BASIC, il est possible de transférer des données vers tous les périphériques en ouvrant des "canaux" vers ceux-ci par l'instruction OPEN : Le cassette, l'écran, l'imprimante ou les lecteurs de disquettes peuvent ainsi être adressés par une même instruction (PRINT#) en modifiant simplement le nom du périphérique dans la syntaxe de OPEN.

La liste des périphériques adressables par le MSX-BASIC n'est pas figée, et peut s'étendre par l'adjonction de cartouches de ROM. Sans extension, les noms de périphériques suivant sont reconnus par le langage :

CAS : Cassette.

CRT : Écran en mode texte.

GRP : Écran en mode graphique.

LPT : Imprimante parallèle.

Si des lecteurs de disquettes sont connectés au système, 2 périphériques supplémentaires correspndant aux 2 lecteurs de disquettes peuvent être adressés :

A : Lecteur principal.

B : Lecteur auxiliaire.

REMARQUES :

— Après chargement du Basic disque livré sur une cartouche enfichable ou implantation de MSX-DOS, le MSX-BASIC est étendu

d'un groupe d'instructions spécialisées dans la gestion des fichiers sur disques. Ces extensions viennent s'ajouter aux instructions présentées dans ce chapitre, et sont regroupées au paragraphe 4.6.

— La gestion de l'écran graphique comme un périphérique permet le mélange de texte et de graphique (voir § 5.1).

### 4.5.2. Les instructions et fonctions de transfert

Lorsque l'ordinateur écrit sur l'écran, il envoie une suite d'octets correspondant aux codes ASCII des caractères affichés. Il en va de même lors d'un transfert de programme vers l'imprimante par LIST ou vers le cassette pour le sauvegarder avec SAVE. Ce principe de fonctionnement permet de comprendre la procédure employée par le MSX-BASIC pour gérer des fichiers : Pour conserver les variables d'un programme, il suffit d'envoyer vers le cassette la suite d'octets les constituant dans l'ordre de leur rangement en mémoire. Ce mode appelé "fichier séquentiel" est utilisé pour tous les transferts de données du langage.

Toutes les instructions et fonctions présentées dans cette section servent à créer, sauvegarder et gérer des fichiers séquentiels. Certaines règles doivent être connues pour pouvoir utiliser ces commandes :

— Dans un fichier séquentiel, les données sont stockées les unes après les autres (séquentiellement) au fur et à mesure de leur arrivée. L'ajout d'informations se fait à partir de la fin du fichier, il n'est pas possible de modifier une donnée se trouvant au milieu du fichier.

— La fin d'un fichier est repérée par un caractère de contrôle réservé, de code ASCII 26 (CTRL/Z).

— Pour retrouver une information en mode séquentiel, il est nécessaire de parcourir tout le fichier du début à la fin, l'accès direct étant impossible.

— Pour chaque fichier pouvant être ouvert, une zone appelée "Buffer" ou "mémoire tampon" est réservée en mémoire. Toute donnée transférée vers ou à partir d'un périphérique transite ensuite par cette zone. Ce transfert s'effectue par blocs : Ce n'est qu'une fois le Buffer rempli que les données sont effectivement envoyées au périphérique adressé.

#### MAXFILES

*Syntaxe* : MAXFILES=/N/

*Application* : Spécifie le nombre de fichiers pouvant être ouvert simultanément en mémoire. /N/ doit être compris entre 0 et 15, et vaut 1 par défaut. Si /N/=0, seules les instructions de sauvegarde et de chargement de programme SAVE et LOAD sont acceptées par le Basic.

REMARQUE : MAXFILES réserve automatiquement un buffer par fichier et diminue la quantité de mémoire disponible.

*Voir aussi* : OPEN.

**OPEN**

*Syntaxe* : OPEN "/Périphérique/[/Nom fichier/]" [FOR/Mode/] AS [#]/N° du fichier/

*Application* : Ouverture d'un fichier séquentiel vers le périphérique spécifié. Pour pouvoir accéder au contenu d'un fichier, il faut au préalable lui affecter un Buffer, définir le mode (lecture ou écriture), et choisir le périphérique adressé.

— /Périphérique/ doit être un des descripteurs accepté par le MSX-BASIC, par exemple CAS : (voir § 4.5.1).

— /Nom fichier/ peut être omis, il sert à caractériser le fichier. On l'utilise surtout lors de sauvegardes sur cassette ou sur disquettes, pour différencier les enregistrements entre eux.

— /Mode/ peut être l'un des trois modes suivant :

OUTPUT Le fichier est ouvert en écriture uniquement. C'est le mode employé pour créer un nouveau fichier.

INPUT Le fichier est ouvert en lecture.

APPEND Le fichier est ouvert en écriture, les données entrées sont ajoutées à la fin du fichier.

— /N° de fichier/ sert à caractériser le fichier pour toutes les instructions d'entrée sortie du MSX-BASIC. Sa valeur maximum est définie par l'instruction MAXFILES. Un fichier ouvert avec un numéro donné lui reste associé jusqu'à sa fermeture par CLOSE.

— Une fois l'OPEN exécuté, cinq instructions permettent de lire et d'écrire dans le fichier :

```
PRINT#  PRINT# USING              En mode écriture

INPUT#  LINE INPUT#  INPUT$(#)    En mode lecture.
```

REMARQUES :

— /Mode/ peut parfois être omis, dans le cas où un seul sens est admis pour un périphérique donné : C'est le cas de **GRP :** et de **LPT :**, sur lesquels on peut écrire uniquement (OUTPUT).

— Le caractère [#] qui précède le numéro du fichier peut également être omis sans provoquer d'erreur.

*Voir aussi* : CLOSE.

***Exemples*** :

```
OPEN "CAS:TEST"  FOR  OUTPUT  AS #1
```

Ouvre un fichier séquentiel en mode écriture sur le cassette. Le numéro 1 sert à accéder au fichier pour les instructions du MSX-BASIC.

```
10 MAXFILES=2
20 OPEN   "LPT:"  AS  #2
```

OPEN ouvre le fichier numéro 2 vers l'imprimante en mode écriture.

**PRINT#**
**PRINT# USING**

*Syntaxes* :

PRINT# /N° du fichier/,/Liste de données/
PRINT# /N° du fichier/,USING/Chaîne/;/Liste de données/

*Application* : PRINT# et son option USING sont identiques à l'instruction PRINT pour un fichier : Les données sont envoyées vers le périphérique spécifié dans OPEN au lieu de l'écran.

REMARQUES :

— PRINT# sans ouverture du fichier adressé provoque une erreur.
— Le fichier doit être ouvert en mode OUTPUT ou APPEND par l'instruction OPEN.
— Il est possible d'utiliser PRINT# exactement comme PRINT en spécifiant l'écran texte comme périphérique dans la syntaxe de OPEN.

*Voir aussi* : PRINT, PRINT USING.

***Exemples*** :

```
10 OPEN "CRT:" FOR OUTPUT AS #1
20 PRINT #1,USING "&    &";"BONJOUR"
```

Écrit sur l'écran à la position du curseur.

```
10 OPEN "LPT:" FOR OUTPUT AS #1
20 PRINT #1,"BONJOUR"
```

Écrit sur l'imprimante.

```
10 OPEN "CAS:TEST" FOR APPEND AS #1
20 PRINT#1,"BONJOUR"
```

Ajoute le mot BONJOUR à la suite du fichier TEST sur cassette.

**Sauvegarde d'un tableau de chaîne :**

```
10 ' CHARGEMENT D'UN TABLEAU DE 10 NOMS SUR K7
15 '
18 ' SAISIE DES NOMS
20 '
25 CLS:DIM F$(10)
30 FOR L=1 TO 10
40 ?:? "NOM No";L;
50 INPUT F$(L)
60 NEXT L
70 ' SAUVEGARDE DU TABLEAU
80 '
90 ?:?"METTEZ LE K7 EN RECORD ET PRESSEZ";
100 ?" UNE TOUCHE"
110 X$=INPUT$(1)
120 OPEN "CAS:NOM" FOR OUTPUT AS #1
130 FOR L=1 TO 10
140 PRINT #1,F$(L)
150 NEXT L
160 CLOSE
```

**Explications :**

— 25 : Le tableau F$ est dimensionné.

— 30-60 : L'utilisateur entre successivement 10 noms.

— 90-110 : Blocage du programme par INPUT$.

— 120 : Ouverture du fichier NOM en écriture.

— 130-150 : Copie du tableau F$ sur cassette. Comme pour PRINT, chaque élément du tableau est séparé du suivant par un retour chariot.

— 160 : Fermeture du fichier.

### INPUT#

*Syntaxe* : INPUT #/N° de fichier/,/Liste de variables/

*Application* : Comme INPUT, cette instruction saisit des données sur le fichier spécifié et les assigne à des variables. Les règles de syntaxe sont identiques, mais aucun point d'interrogation n'est affiché par INPUT# : Les données sont saisies dans un buffer, pas au clavier !

Dans le cas des variables numériques, le premier caractère rencontré qui n'est pas un espace, un retour chariot ou un saut de ligne est considéré comme le début d'un nombre. Le nombre se termine par une virgule ou un de ces trois séparateurs.

Pour les chaînes de caractères, le début d'une chaîne est repéré de la même manière que pour les nombres, ou par un guillemet. Dans le premier cas, la chaîne est considérée comme terminée après rencontre d'une virgule, d'un retour chariot ou d'un saut de ligne, ou encore après saisie de 255 caractères. Dans le deuxième cas, la chaîne est saisie jusqu'à rencontre de guillemets, quelque soit son contenu.

REMARQUE: Lors d'une entrée de données par INPUT#, la rencontre du caractère CTRL/Z indiquant la fin de fichier interrompt le déroulement de la saisie.

*Voir aussi*: INPUT, LINE INPUT#.

***Exemple***: Le tableau de 10 noms F$ sauvegardé par PRINT# Peut être rechargé par INPUT#:

```
10 ' LECTURE D'UN TABLEAU DE 10 NOMS SUR K7
20 '
30 CLS:DIM F$(10)
40 ? "METTEZ LE K7 EN LECTURE"
50 OPEN "CAS:NOM" FOR INPUT AS #1
60 FOR L=1 TO 10
70 INPUT #1,F$(L)
80 NEXT L
90 CLOSE
100 '
110 ' AFFICHAGE DU TABLEAU
120 '
130 FOR L=1 TO 10
140 ? "NOM No";L;"=";F$(L)
150 ?
160 NEXT L
```

**Explications:**

— 30: Dimensionne F$.

— 50: Ouverture du fichier NOM en lecture.

— 60-80: Saisie des noms par INPUT# (le retour chariot sert de séparateur et impose les passages à la ligne).

— 90: Fermeture du fichier.

— 130-160: Affichage du tableau F$.

## LINE INPUT#

*Syntaxe*: LINE INPUT #/N° fichier/,/Variable chaîne/

*Application* : Permet de lire une chaîne de caractères à partir d'un fichier séquentiel, sans tenir compte des séparateurs excepté le retour chariot (code 13). Comme LINE INPUT, jusqu'à 254 caractères peuvent être saisis par l'instruction.

REMARQUE : On utilise souvent LINE INPUT# pour saisir dans une seule variable chaîne les constituants d'un fichier séparés par une virgule, que INPUT# traite comme un séparateur.

*Voir aussi*: LINE INPUT, INPUT#.

***Exemple*** :

```
LINE INPUT #1,A$
```

## INPUT$(#)

*Syntaxe*: INPUT$ (N,[#]/N° fichier/)

*Application*: Comme INPUT$, cette commande retourne une chaîne de N caractères saisie sur le fichier spécifié. Tous les caractères sont acceptés, y compris le retour chariot (code 13).

REMARQUE :

— N doit être compris entre 0 et 255.

— Le symbole # peut être omis.

*Voir aussi*: INPUT$.

***Exemple*** :

```
X$=INPUT$  (25,#1)  saisit  25
```

## CLOSE

*Syntaxe*: CLOSE [[#]/N° fichier/[,/N° fichier/]]

*Application*: Ferme tous les fichiers nommés et libère les mémoires tampon qui leur sont associées. Si aucun argument ne suit CLOSE, tous les fichiers sont fermés simultanément.

REMARQUE : L'instruction CLOSE doit impérativement être employée chaque fois qu'un fichier ne doit plus être utilisé, car elle transfère toutes les données restant dans le buffer vers le périphérique spécifié et fait suivre le fichier du code de fin 26 (CTRL/Z).

*Voir aussi*: OPEN.

**EOF**

*Syntaxe*: EOF (/N° fichier/)

*Application*: La fonction retourne −1 si la fin d'un fichier séquentiel est atteinte lors d'une lecture par INPUT#, LINE INPUT# ou INPUT$ (#).

REMARQUE: EOF permet d'éviter un message d'erreur en tentant une lecture dans un fichier dont la fin est atteinte.

***Exemple***: saisit 25 caractères dans le fichier spécifié.

```
10 OPEN "CAS:TEST" FOR INPUT AS #1
20 INPUT #1,A$
30 IF EOF(1)=-1 THEN END
40 GOTO 20
```

### 4.5.3. Un fichier d'adresses

Ce programme de fichier utilise les instructions présentées au chapitre précédent pour gérer un carnet d'adresses en séquentiel sur le cassette. Quatre fonctions sont prévues au menu principal: Création d'une fiche, Visualisation d'une fiche, Sauvegarde et chargement d'un fichier. C'est évidemment une gestion de fichier assez élémentaire (en particulier les fonctions modification/destruction de fiche et visualisation de tout le fichier en sont absentes), mais le programme est parfaitement utilisable tel quel. Il est suivi d'explications détaillées devant permettre au lecteur qui le désire d'y ajouter des possibilités du type tri ou impression:

```
 10 ' FICHIER D'ADRESSES
 20 '
 30 SCREEN 0:COLOR 15,1,4:KEY OFF
 40 L=1:DIM F$(600)
 50 GOTO 550
 60 '
 70 ' CREATION
 80 '
 90 CLS:?:?"CREATION FICHE No"INT(L/4+1):?:?
100 INPUT"NOM ";F$(L):L=L+1:?
110 INPUT"PRENOM ";F$(L):L=L+1:?
120 INPUT"ADRESSE ";F$(L):L=L+1:?
130 INPUT"TELEPHONE ";F$(L):L=L+1:?:?
140 INPUT "VALIDATION (O/N)";R$:IF R$="N" THEN L=L-4:GOTO 70
    ELSE RETURN
150 '
160 ' VISUALISATION
170 '
180 IF L<2 THEN RETURN
190 CLS:?:?"VISUALISATION"
200 LOCATE 0,6:INPUT "NOM RECHERCHE";N$
210 N=1
220 IF LEFT$(F$(N),LEN(N$))=N$ THEN 240
```

```
230 N=N+4:IF N>L THEN LOCATE 0,15:?"FICHE NON TROUVEE":GOTO 280
    ELSE 220
240 CLS:?:?"FICHE No"INT(N/4+1);:LOCATE 0,6
250 ?"NOM:":? F$(N)" "F$(N+1):?
260 ?"ADRESSE:":?F$(N+2):?
270 ?"TELEPHONE:":?F$(N+3)
280 LOCATE 1,20:INPUT"UNE AUTRE FICHE (O/N) ";R$:IF R$="O" THEN
    GOTO 160 ELSE RETURN
290 '
300 ' SAUVEGARDE
310 '
320 IF L<2 THEN RETURN
330 CLS:LOCATE 0,10:?"METTEZ LE K7 EN RECORD ET PRESSEZ":?"UNE
    TOUCHE"
340 X$=INPUT$(1)

350 F$(0)=STR$(L)
360 OPEN "CAS:ADRESSE" FOR OUTPUT AS #1
370 FOR I=0 TO L
380 IF F$(I)="" THEN F$(I)=" "
390 PRINT #1,F$(I)
400 NEXT I
410 CLOSE
420 RETURN
430 '
440 ' LECTURE
450 '
460 CLS:LOCATE 0,10:?"METTEZ LE K7 EN LECTURE"
470 OPEN "CAS:ADRESSE" FOR INPUT AS #1
480 INPUT #1,F$(0):L=VAL (F$(0))
490 FOR I=1 TO L
500 INPUT #1, F$(I)
510 NEXT I
520 CLOSE
530 RETURN
540 '
550 ' MENU
560 '
570 CLS:LOCATE 14,1:?"MENU":?:?
580 ?"1  CREER UNE FICHE"
590 ?"2  VISUALISER UNE FICHE"
600 ?"3  SAUVEGARDER LE FICHIER"
610 ?"4  CHARGER UN FICHIER"
620 ?:?:?:INPUT"VOTRE OPTION ";R
630 ON R GOSUB 70,160,300,440
640 GOTO 550
```

REMARQUE: Il suffit de remplacer CAS: par A: dans les OPEN des lignes 360 et 470 pour créer le fichier d'adresses sur disquettes.

**Explications :**

Comme on peut le voir clairement d'après le listing, quatre modules remplissant les fonctions principales ont été utilisés, plus le module "MENU" qui les appelle selon les besoins.

La structure du fichier est assez simple: Chaque fiche occupe 4 éléments du tableau F$( ), le premier est le nom, le deuxième le prénom, le troisième l'adresse et le dernier le téléphone. Pour accéder à une fiche, il suffit de démarrer à 1 puis d'ajouter 4 à chaque fois pour trouver le nom suivant.

Voici les variables utilisées:

| | |
|---|---|
| F$(600) | Tableau contenant les fiches (jusqu'à 150). |
| R$ | Réponse de l'utilisateur. |
| N$ | Nom de la fiche recherchée. |
| L | Nombre d'éléments de F$ utilisés. Le nombre de fiches s'obtient par L/4+1. |
| R | Réponse de l'utilisateur dans le menu. |
| N,I | Compteurs de boucles. |

**Structure du programme:**

— 70,130: Module création. Chaque rubrique est saisie dans un élément de F$, L étant incrémenté de 1 à chaque fois.

— 140: L'utilisateur a la possibilité de saisir la fiche à nouveau s'il a commis une erreur (attention INPUT n'accepte pas les virgules dans les chaînes de caractères).

— 160,280: Module de visualisation. La recherche d'une fiche se fait sur le nom: La variable N démarre à 1, puis compare F$(N) avec N$. S'ils sont différents, on ajoute 4 à N et on recommence.

— 230: Si N dépasse L, c'est que la fiche n'existe pas: Un message approprié est affiché.

— 240,270: Les quatre éléments de F$ composant la fiche sont affichés.

— 280: Que la fiche ait été trouvée ou non, on demande si l'utilisateur veut continuer ou retourner au menu.

— 300,420: Module sauvegarde du tableau F$. L'élément F$(0), non utilisé pour le fichier, est chargé avec la valeur de L avant la sauvegarde (ligne 350): Ainsi la taille du fichier est sauvegardée avec lui.

— 440,530: Module de relecture de F$. On lit d'abord l'élément F$(0), qui contient la taille du fichier L: Cette astuce permet de connaître le nombre d'éléments sauvegardés sur cassette. On aurait également pu écrire, en utilisant EOF:

```
470 OPEN "CAS:ADRESSE" FOR INPUT AS #1
480 I=0
500 INPUT #1, F$(I)
510 IF EOF<>-1 THEN I=I+1:GOTO 500
520 CLOSE
530 RETURN
```

— 550,640 : Menu principal. Remarquez la ligne 630, qui effectue le branchement vers les différents modules en fonction de la valeur de R.

## 4.6. MSX-DOS : LE SYSTÈME D'EXPLOITATION DE DISQUETTES DE MSX

### 4.6.1. Généralités

En l'absence de lecteurs de disquettes, la gestion des divers périphériques (lecteur de cassettes, vidéo, imprimante) est assurée dans sa totalité par le Basic MSX en ROM, qui assure ainsi les tâches normalement dévolues au système d'exploitation.

Cela n'est plus possible lorsqu'on utilise des lecteurs de disquettes, car alors d'autres langages doivent pouvoir être chargés à partir du disque : Le Basic en ROM perd alors le contrôle de la machine, et du même coup, la supervision des diverses entrées-sorties et programmes exécutés.

Ce contrôle, qui devient d'ailleurs considérablement plus complexe lorsque des disquettes doivent être gérées, est assumé par un programme spécialisé, que l'on nomme **système d'exploitation**. Il s'agit de MSX-DOS (DOS pour Disk Operating System) dans notre cas, un système qui présente de nombreux points communs avec son aîné MS-DOS, très utilisé sur les micro-ordinateurs professionnels 16 bits.

De plus, de nouvelles instructions et commandes sont rendues nécessaires par la présence du disque, aussi bien celles qui lui sont spécifiques (copie et formattage de disquettes, utilitaires divers, etc...) que celles qu'il permet d'ajouter à un langage évolué. Elles concernent principalement (mais pas uniquement) la gestion des **fichiers à accès direct** (random files), qui nécessite de nombreuses instructions supplémentaires ; celles-ci sont décrites ci-après, en 4.6.5.

Deux versions différentes de MSX-DOS sont disponibles :

— **Une version sur cartouche**, dans laquelle le système est incorporé au Basic-Disque étendu. Pratiquement, l'utilisateur se trouve toujours sous Basic exactement comme à l'accoutumée, avec la différence qu'une seule commande système est accessible, grâce à une syntaxe spéciale, et qu'un certain nombre d'instructions et fonctions Basic supplémentaires sont disponibles.

  L'inconvénient de cette version est qu'il n'est pas possible de quitter le Basic, et donc de charger d'autres langages ou d'intervenir direc-

tement au niveau du système. Son avantage est qu'elle peut fonctionner avec moins de 64 Ko de mémoire vive.

— **Une version disque**, où le système est résident sur une disquette, ainsi que le Basic étendu ou tout autre langage.

  Cette version est la plus proche des systèmes professionnels (MS-DOS 1.1), et permet à l'utilisateur de se rendre directement sous MSX-DOS pour la gestion de ses disquettes (copie, formattage, etc.), qui est assurée par de nombreuses fonctions, ou pour charger d'autres langages ou utilitaires. Par contre, cet accès au système n'est possible que si la machine dispose de la totalité de la mémoire vive (64 Ko).

### 4.6.2. Caractéristiques communes aux deux versions du DOS

MSX-DOS est en fait constitué de plusieurs programmes, chacun chargé d'une activité spécifique.

Ceux-ci assurent en permanence le contrôle de l'ordinateur, et supervisent le chargement et l'exécution des programmes (dans ce contexte, un langage comme le Basic est lui-même considéré comme un programme), ainsi que l'acheminement des données exigé par ces derniers vers les diverses unités périphériques, disque compris.

Le DOS assure en totalité la gestion de l'espace-disque, c'est-à-dire qu'il tient à jour le **répertoire** de la disquette, ainsi que l'enregistrement ou la lecture des **fichiers** aux emplacements appropriés.

**Un fichier** est un ensemble de données, enregistrées sur le disque sous un même nom. Pour le système, un programme est un fichier au même titre qu'un texte, ou un tableau de variables sauvegardé par le Basic.

Les *noms de fichier* répondent à la syntaxe suivante :

U :[nom fichier].[extension]

Où :

| | |
|---|---|
| U : | Est le lecteur où se trouve le fichier, le premier étant le A :, le deuxième B :, etc... |
| nom fichier | Est un mot de 8 caractères alphanumériques maximum, choisi par l'utilisateur, qui définit le fichier. |
| extension | Est une extension de trois caractères alphanumériques maximum, qui permet de grouper les fichiers par catégories, etc... Celle-ci n'est pas obligatoire, mais un point doit la séparer du nom du fichier lorsqu'elle est utilisée. |

REMARQUE : Certaines extension sont attribuées par défaut à des fichiers particuliers. Par exemple, l'extension .BAS sera attribuée automatiquement lors de la sauvegarde d'un programme Basic, si l'utilisateur n'en a pas spécifié une autre.

Deux **caractères joker** peuvent être utilisés dans les noms de fichiers, permettant de se reférer à tout un groupe de fichiers.

— Le caractère ? (point d'interrogation) se réfère à n'importe quel caractère occupant la même place. Ainsi, A??Z.TEX pourra correspondre à AAAZ.TEX, ou ABCZ.TEX, ou A01Z.TEX, etc...

— Le caractère * (étoile) se réfère à n'importe quel groupe de caractères. Ainsi, *.BAS se réfèrera à tous les fichiers portant l'extension .BAS, quel que soit leur nom.

Le **répertoire**, ou directory, est la liste de tous les fichiers présents sur une disquette. Il est constamment tenu à jour par MSX-DOS, et peut être consulté par l'intermédiaire du Basic (instruction FILES), ou directement sous DOS dans la version disque du système.

Il faut savoir enfin que la gestion de l'espace disque est entièrement transparente à l'utilisateur, c'est-à-dire que celui-ci n'a pas à se préoccuper de l'endroit physique où sont placés les fichiers, comme sur une cassette. Le DOS sait en permanence où enregistrer et lire les données, seuls les noms de fichiers et les commandes à exécuter doivent être fournis par l'utilisateur.

### 4.6.3. MSX-DOS sur cartouche (ou sur disque, avec moins de 64 K)

Dans la mesure où dans cette version, toutes les opérations s'effectuent sous Basic, une syntaxe spéciale est nécessaire pour accéder à l'unique commande du système. Il en va de même avec le DOS sur disque, lorsque moins de 64 Ko de RAM sont disponibles.

Pour être interprétée comme telle, une commande-système entrée sous Basic doit être précédée du caractère _ (trait de soulignement), ou de l'instruction CALL.

Cette commande est :

#### FORMAT

*Syntaxe* : _FORMAT U :

*Application* : Formatte une disquette vierge. Cette opération, qui consiste à inscrire des repères magnétiques figurant les pistes et secteurs, doit être effectuée sur toute disquette neuve avant la première utilisation.

***Exemple*** : CALL FORMAT A : ou FORMAT A : formatte la disquette se trouvant dans le lecteur A.

### 4.6.4. MSX-DOS sur disque, avec 64 Ko de mémoire

Dans le cas où l'on dispose du MSX-DOS sur disque et de 64 Koctets de mémoire vive, le démarrage de l'ordinateur s'effectue sur la disquette-système. Dès que le système est chargé en mémoire, MSX-DOS affiche son indicatif :

**A**] indiquant qu'il est prêt à recevoir une commande. Un programme spécial, nommé **COMMAND.COM**, effectue l'interprétation et l'exécution de vos commandes. Il n'est plus nécessaire de les faire précéder de _ ou de l'instruction CALL, comme lorsque l'on y accède par l'intermédiaire du Basic. Mais vous pouvez aussi, bien entendu, charger le Basic si vous le désirez en tapant simplement **MSXBASIC** suivi de RETURN.

Le lecteur sur lequel se trouve MSX-DOS au moment du démarrage devient le **lecteur par défaut**. C'est lui qui apparaît dans l'indicatif, et son nom n'a pas besoin d'être spécifié dans les commandes. Il est très facile de changer de lecteur par défaut : Pour passer de A en B, par exemple, tapez : B : puis RETURN. Maintenant, l'écran affiche : B]. B est devenu le nouveau lecteur par défaut.

#### L'éditeur de COMMAND.COM

Afin de vous aider à entrer et modifier vos commandes, COMMAND.-COM dispose d'un petit éditeur simplifié. Lorsque vous tapez une ligne après l'indicatif A] puis la touche RETURN, celle-ci est interprétée et exécutée par MSX-DOS. Dans le même temps, elle est placée dans une mémoire-tampon appelée BUFFER.

Plusieurs touches vous permettent de recopier sélectivement le contenu du buffer à l'écran, comme si vous aviez retapé le texte manuellement. Cela permet de répéter plusieurs fois une ligne, en la modifiant le cas échéant. Voici les touches d'édition :

| *Touche* | *Fonction* |
|---|---|
| → | Recopie un caractère du buffer à l'écran. |
| DEL | Saute (ne copie pas à l'écran) un caractère du buffer. |
| SELECT | Suivi d'un caractère, recopie à l'écran le contenu du buffer jusqu'au caractère spécifié. |
| CLS | Suivi d'un caractère, saute (ne copie pas à l'écran) tout le contenu du buffer jusqu'au caractère spécifié. |
| ↓ | Copie à l'écran tout le restant du buffer. |
| ESC | Annule la ligne courante. |
| HOME | Recommence l'édition de la ligne courante. |
| ←, BS | Recule d'un caractère dans le buffer. |
| INS | Passe du mode surcharge au mode insertion de caractères, et vice-versa. |

***Exemple***: Supposons que vous ayez tapé COPY A:PROGRAM.BAS B:TOTO.*

```
- Tapez: SELECT puis .
  L'écran affiche: A]COPY A:PROGRAM

- Tapez: INS, puis 1
  L'écran affiche: A]COPY A:PROGRAM1

- Tapez: → 3 fois
  L'écran affiche: A]COPY A:PROGRAM1.BA

- Tapez: DEL, puis ↓
  L'écran affiche: A]COPY A:PROGRAM1.BAS B:TOTO.*
```

**Les fichiers BATCH ou fichiers de commandes**

Les Fichiers **BATCH** sont des fichiers spéciaux portant l'extension .BAT, qui peuvent contenir une ou plusieurs lignes de texte.

Chaque ligne est en fait un ligne de commandes, qui est interprétée et exécutée par MSX-DOS exactemnt comme si elle avait été entrée au clavier. Vous pouvez créer vos propres fichiers BATCH à l'aide d'un éditeur de textes, par le Basic sous forme de fichiers séquentiels, ou en suivant la méthode de notre exemple ci-dessous.

Un fichier BATCH est exécuté simplement en tapant son nom (sans extension), suivi de RETURN. Cela n'est possible bien entendu que lorsque l'on se trouve sous DOS, donc que l'indicatif A] apparaît à l'écran: Il n'est pas possible d'exécuter un fichier Batch lorsqu'on se trouve sous Basic.

Lors du démarrage du système , MSX-DOS recherche sur la disquette la présence d'un fichier BATCH spécial, nommé **AUTOEXEC.BAT**. S'il existe, celui-ci est exécuté, sinon le programme Basic **AUTOEXEC.BAS** est recherché également. S'il n'est pas trouvé non plus, l'indicatif A] est affiché.

***Exemple*** : Création d'un fichier BATCH.

Tapez la séquence suivante (A] est l'indicatif affiché par MSX-DOS) :

```
A] COPY CON AUTOEXEC.BAT     RETURN
   DATE                      RETURN
   MSXBASIC                  RETURN
```

Tapez CTRL et Z en même temps, puis la touche RETURN.

Vous venez de créer sur votre disquette-système un fichier AUTOEXEC.-BAT. Désormais, à chaque démarrage du système, MSX-DOS vous demandera la date, puis chargera automatiquement le Basic.

### Les commandes de MSX-DOS sur disque

Ces commandes, comme nous l'avons dit, sont très proches de celles que l'on trouve sur MS-DOS 1.1. En voici la liste :

## FORMAT

*Syntaxe* : FORMAT [/S] U :

*Application* : Comme pour MSX-DOS sur cartouche, cette commande formatte une disquette vierge. Dans ce cas, FORMAT n'a évidemment pas besoin d'être précédé de _ ou de CALL.

REMARQUE : L'option /S permet de transférer MSX-DOS sur la disquette formattée, et de créer ainsi une nouvelle disquette-système.

***Exemple*** : FORMAT A : formatte la disquette se trouvant dans le lecteur A.

## COPY

*Syntaxe* : COPY /U :fichier source//U :fichier dest/

*Application* : Copie "fichier source" sous le nom "fichier dest". Si la copie s'effectue sur un lecteur différent du premier, et que [fichier dest] est omis, le fichier sera copié avec le même nom.

***Exemples*** :

COPY A :PROG.BAS B : copie le fichier spécifié sur le lecteur B :, avec le même nom.

COPY PROG.BAS COPIE.* recopie le fichier spécifié sur le même lecteur, sous le nom COPIE.BAS (en gardant la même extension).

**DISKCOPY**

*Syntaxe*: DISKCOPY U1: U2:

*Application*: Copie une disquette entière, du premier lecteur spécifié vers le deuxième. La copie effectuée est un double parfait de l'original.

REMARQUE: Si un seul lecteur est disponible, une séquence intéractive vous demande d'y introduire tour à tour l'une ou l'autre disquette.

***Exemple***: DISKCOPY A: B: recopie la disquette se trouvant dans le lecteur A: sur celle se trouvant dans le lecteur B:.

**DIR**

*Syntaxe*: DIR U:

*Application*: Affiche à l'écran la liste des fichiers se trouvant sur le lecteur spécifié. Si U: est omis, DIR renvoie la liste des fichiers du lecteur par défaut.

**DELETE**

*Syntaxe*: DELETE U:/nom fichier/

*Application*: Détruit le fichier spécifié. Des caractères-joker peuvent être utilisés dans le nom, pour détruire tout un groupe de fichiers. Cette commande est équivalente à l'instruction Basic KILL.

***Exemple***: DELETE B:*.* détruit tous les fichiers du lecteur B:

**RENAME**

*Syntaxe*: RENAME U:/nom fichier1//nom fichier2/

*Application*: Change le nom de [nom fichier1], qui devient [nom fichier2]. Des caractères-joker peuvent être utilisés, pour renommer tout un groupe de fichiers. Cette commande est équivalente à l'instruction Basic NAME... AS.

***Exemple***: RENAME *.BAS *.PRG change l'extension de tous les fichiers ayant l'extension .BAS en .PRG, en conservant leur nom.

**TYPE**

*Syntaxe*: TYPE U:/nom fichier/

*Application*: Affiche à l'écran le contenu du fichier spécifié (ne fonctionne que pour les fichiers séquentiels, ou fichiers texte).

### DATE

*Syntaxe*: DATE [mm/jj/aa]

*Application*: Permet de connaître la date mémorisée par le système, ou de la remettre à jour. Le jour et le mois doivent être des nombres à deux chiffres, l'année peut avoir quatre chiffres.

***Exemple***: DATE 8/21/84 initialise la date au 21 Août 1984.

### TIME

*Syntaxe*: TIME [hh :mm :ss]

*Application*: Permet de connaître l'heure mémorisée par le système, ou de la remettre à jour.

REMARQUE: L'heure est tenue à jour en permanence, et TIME renverra l'heure réelle si celle-ci a été correctement initialisée au démarrage du système.

## 4.6.5. Le Basic disque étendu

Ce Basic, livré avec MSX-DOS, supporte toutes les instructions du Basic standard plus toutes celles gérant les fichiers à accès direct, ainsi que certaines instructions spécifiques à l'utilisation des disquettes.

Toutes les commandes gérant l'enregistrement et la sauvegarde de programmes, et les fichiers séquentiels, fonctionnent également sur disque. Ces instructions et fonctions sont décrites en détail aux paragraphes 4.3 et 4.5, mais nous les rappelons ici pour mémoire avec leur syntaxe "disque".

Les règles d'appellation des fichiers sont les mêmes que celles données plus haut, à savoir **U :nfichier.ext**. Cet ensemble de trois éléments doit toujours être inclus entre guillemets sous BASIC, ce qui n'est pas le cas sous MSX-DOS.

Signalons enfin que les caractères-joker (? et *) sont tout à fait utilisables sous Basic, du moins dans les commandes pouvant faire référence à un groupe de fichiers plutôt qu'à un seul comme FILES, KILL ou NAME.

***Commandes existant déjà en Basic standard***

**Instructions de gestions des programmes**

**SAVE**

*Syntaxe*: SAVE "U:nfichier.ext" [,A]

*Application*: Sauvegarde sur disque le programme en mémoire, sous le nom spécifié. Si l'option ,A est employée, le programme est sauvegardé sous forme de texte (ASCII), et peut être listé directement sous MSX-DOS avec la commande TYPE.

**LOAD**

*Syntaxe*: LOAD "U:nfichier.ext" [,R]

*Application*: Charge en mémoire le programme spécifié. Son exécution commence directement si l'option ,R est utilisée.

**BSAVE**

*Syntaxe*: "U:nfichier.ext",A1,A2 [,A3]

*Application*: Sauvegarde sur disque la zone-mémoire comprise entre les adresses A1 et A2, sous le nom de fichier spécifié.

**BLOAD**

*Syntaxe*: BLOAD "U:nfichier.ext" [,R]

*Application*: Charge le bloc-mémoire sauvegardé sur disque sous le nom spécifié.

**MERGE**

*Syntaxe*: MERGE "U:nfichier.ext"

*Application*: Incorpore le programme spécifié au programme déjà présent en mémoire, sans détruire ce dernier. Le programme à incorporer doit avoir été sauvegardé sous forme ASCII, avec l'option ,A de SAVE.

**RUN**

*Syntaxe*: RUN ["U:nfichier.ext"]

*Application*: Lorsqu'un nom de fichier est spécifié, le programme est chargé à partir du disque puis exécuté.

**Instructions et fonctions gérant les fichiers séquentiels**

Nous avons déjà vu les fichiers séquentiels appliqués aux cassettes, au paragraphe 4.5. Les mêmes instructions commandes fonctionnant sur cassette sont également valable pour le disque :

*A) Instructions*

### OPEN

*Syntaxe*: OPEN "U:nfichier.ext" FOR /mode/ AS #N

*Application*: Ouvre le fichier spécifié, sous le numéro N.

INPUT, le fichier est ouvert en lecture. Un message d'erreur est affiché si le fichier n'existe pas.

OUTPUT, le fichier est ouvert en écriture. S'il n'existe pas, il est créé, sinon l'ancien fichier est surchargé.

APPEND comme OUTPUT, mais l'écriture commence à la fin du fichier si celui-ci existe déjà.

### CLOSE

*Syntaxe*: CLOSE #N

*Application*: Ferme le fichier ouvert sous le numéro N. Le buffer est libéré, et le répertoire de la disquette mis à jour.

### PRINT # et PRINT # USING

*Syntaxe*: PRINT #N,A$ et PRINT #N USING "/format/",A$

*Application*: Écrit la chaîne A$ dans le fichier séquentiel ouvert sous le numéro N. PRINT USING permet de formatter les données à l'écriture.

### INPUT # et LINE INPUT #

*Syntaxe*: INPUT #N,A$ et LINE INPUT #N,A$

*Application*: Lit l'enregistrement courant dans le fichier ouvert sous le numéro N, et l'attribue à la variable A$. LINE INPUT permet de ne pas tenir compte des séparateurs tels que virgules, etc...

*B) Fonctions*

### INPUT$

*Syntaxe*: A$=INPUT$(X,#N)

*Application*: Lit X caractères de l'enregistrement courant du fichier ouvert sous le numéro N, et les attribue à A$.

## EOF

*Syntaxe*: EOF(N)

*Application*: Cette fonction retourne −1 si ce dernier enregistrement du fichier ouvert sous le numéro N a été lu, et retourne 0 dans tous les autres cas.

## LOC

*Syntaxe*: LOC(N)

*Application*: Retourne, pour le fichier ouvert sous le numéro N, le nombre de secteurs lus ou écrits depuis son ouverture par OPEN. Chaque secteur compte 128 octets. Cette instruction n'est pas disponible en MSX-BASIC non étendu.

### *Commandes spécifiques du Basic disque étendu*

### Instructions d'intérêt général

## FILES

*Syntaxe*: FILES ["U :nfichier.ext"]

*Application*: Affiche le nom du (ou des) fichier(s) spécifié(s). Si un seul fichier est donné, seul son nom sera affiché. Pour afficher une sélection de fichiers, il faut utiliser les caractères-joker ? et *. FILES sans paramètres retourne la liste de tous les fichiers du lecteur par défaut.

***Exemples***:

FILES"B:*.*" donne la liste de tous les fichiers du lecteur B.

FILES"*.BAS" retourne tous les fichiers ayant l'extension .BAS.

## KILL

*Syntaxe*: KILL "U :nfichier.ext"

*Application*: Détruit le fichier spécifié. Cette commande est équivalente au DELETE sous MSX-DOS. Un fichier ouvert par OPEN ne peut être détruit, il faut d'abord le fermer par CLOSE.

## NAME

*Syntaxe*: NAME "U :nfichier.ext" AS "nouveau nom"

*Application*: Change le nom du fichier spécifié en "nouveau nom". Cette instruction est équivalente au RENAME de MSX-DOS.

***Exemple***: NAME "PROGRAM.BAS" AS "PROG1.*" change le nom du ficher PROGRAM.BAS en PROG1.BAS.

## _SYSTEM ou CALL SYSTEM

*Application* : Cette commande provoque l'abandon du Basic, et le retour sous MSX-DOS. S'agissant d'une commande-système, elle doit être précédée de _ ou de CALL.

### *Les fichiers à accès direct*

Les fichiers à accès direct, ou fichiers random, se distinguent des fichiers séquentiels en ce qu'ils permettent la lecture ou la modification de n'importe quel enregistrement, en l'appelant par son numéro d'ordre.

En contrepartie, ils sont plus lourds à mettre en œuvre, au niveau de la programmation, que les fichiers séquentiels. Leur usage se révèle cependant indispensable pour toute application "sérieuse", comme gestion de fichiers clients, carnets d'adresses, stock, etc, où il est indispensable de pouvoir mettre à jour des données individuellement sans avoir à toucher au restant du fichier.

La liste des divers commandes spécifiques aux fichiers random vous est donnée à titre de référence, nous verrons ensuite leur utilisation à l'aide d'un exemple concret.

*A) Instructions*

## OPEN

*Syntaxe* : OPEN "U :nfichier.ext" AS #N [LEN=/longueur/]

*Application* : Ouvre un fichier en mode accès direct sous le nom spécifdié, et lui réserve un buffer (mémoire tampon), de la taille spécifiée dans /longueur/. Celle-ci est exprimée en octets, et dépend de la nature et de la longueur des données, que l'on veut enregistrer dans le fichier. Si ce paramètre est omis, 128 octets sont réservés par défaut. Si le fichier n'existait pas précédemment, il est créé.

## CLOSE

*Syntaxe* : CLOSE #N

*Application* : Ferme le fichier ouvert sous le numéro N.

## FIELD

*Syntaxe* : FIELD #N,X AS C1$,Y AS C2$,... etc

*Application* : Dans un fichier à accès direct, chaque enregistrement peut être divisé arbitrairement en un nombre quelconque de "champs», chaque champ contenant une variable. Il est par contre nécessaire de

définir au préalable, avant tout accès, la taille, en octets, qu'occupera chaque champ de l'enregistrement. Une variable-chaîne réservée est également attribuée à chaque champ, pour permettre d'y accéder ; cette variable ne peut plus être utilisée ailleurs dans le programme.

L'instruction FIELD effectue cette opération. Le premier champ sera accessible par la variable réservée C1$, et aura une longueur de X octets ; le deuxième se nommera C2$, et aura Y octets de long, etc... (les noms C1$, C2$, etc. sont totalement arbitraires). La somme de la taille de tous les champs (X+Y+...+Xn) ne doit pas être supérieure à la taille du buffer, définie dans OPEN par le paramètre /longueur/.

## LSET et RSET

*Syntaxe* : LSET C1$=A$ et RSET C1$=A$

*Application* : Comme les variables réservées ne peuvent plus être utilisées par ailleurs dans le programme, LSET et RSET sont utilisés pour y transférer les données que l'on veut écrire dans le fichier. LSET cadre à gauche la chaîne A$ dans C1$, et RSET la cadre à droite.

***Exemple*** : Supposons que A$ est égal à "DUPONT", et que le champ C1$ a une longueur de 20 octets, définis par FIELD.

— Si l'on écrit LSET C1$=A$, C1$ sera égal à "DUPONT              "
— Si l'on écrit RSET C1$=A$, C1$ sera égal à "              DUPONT"

## GET

*Syntaxe* : GET #N,X

*Application* : Lit l'enregistrement numéro X, dans le fichier ouvert sous le numéro N. Les données sont transférées sur disque, dans les variables réservées définies par FIELD.

## PUT

*Syntaxe* : PUT #N,X

*Application* : Écrit sur disque, dans le fichier ouvert sous le numéro N et dans l'enregistrement X, le contenu des variables réservées par FIELD. Les données à enregistrer auront été placées au préalable dans ces variables réservées.

*B) Fonctions*

### MKI$, MKS$, MKD$

*Syntaxe*: LSET C1$=MKx$(A)
ou RSET C1$=MKx$(A)
Avec MKx$=MKI$, ou MKS$, ou MKD$

*Application*: Un enregistrement d'un fichier à accès direct ne peut contenir que des chaînes de caractères, comme nous l'avons vu dans FIELD. Pour stocker des variables numériques, il faut donc utiliser une conversion. C'est là l'utilité de ces fonctions:

— MKI$ convertit un entier en une chaîne de 2 octets.

— MKS$ convertit un réel simple précision en chaîne de 4 octets.

— MKD$ convertit un réel double précison en chaîne de 8 octets.

### CVI, CVS, CVD

*Syntaxe*: A=CV×(C1$)
avec CVx=CVI, ou CVS, ou CVD

*Application*: Ces fonctions remplissent le rôle inverse de MKI$, MKS$, etc., à savoir qu'elles reconvertissent en nombres les champs d'un enregistement relu dans le fichier.

— CVI reconvertit en entier une chaîne de 2 octets.

— CVS reconvertit en réel simple précision une chaîne de 4 octets.

— CVD reconvertit en réel double précison une chaîne de 8 octets.

REMARQUE: Pour signifier quelque chose, les chaînes reconverties en nombres par CVI, etc... doivent évidemment provenir de nombres eux-mêmes transformés en chaînes par MKI$, etc. au moment de l'écriture de l'enregistrement.

### LOC

*Syntaxe*: LOC(N)

*Application*: Cette fonction retourne le numéro du dernier enregistrement lu par GET ou écrit par PUT, dans le fichier ouvert sous le numéro N.

### LOF

*Syntaxe*: LOF(N)

*Application*: Retourne la longueur, en octets, du fichier ouvert sous le numéro N. Connaissant la longueur de chaque enregistrement (LE) que l'on doit normalement spécifier dans OPEN, il est très facile de retrouver le nombre d'enregistrements (NE) de ce fichier, par: NE=LOF(N)/LE.

### *Création et utilisation d'un fichier à accès direct*

Supposns que nous voulions créer et exploiter un fichier à accès direct, que nous nommerons ARTICLES.FCH. Chaque enregistrement de ce fichier aura 28 octets de long, distribués en quatre "champs" qui seront respectivement le code article (CODE$), la désignation (NOM$), le prix (PRIX$) et la quantité en stock (NOMBRE$) :

— 2 octets seront réservés pour CODE$, qui stocke un entier.
— 20 octets pour NOM$, qui stocke une chaîne de caractères.
— 4 octets pour PRIX$, qui stocke un réel simple précison.
— 2 octets enfin pour NOMBRE$, qui représente aussi un nombre entier.

Les étapes suivantes seront nécessaires à la création du fichier :

**1.** Ouvrir le fichier par OPEN :

```
OPEN "ARTICLES.FCH" AS #1 LEN=28
```

**2.** Définir les champs de l'enregistrement, et les variables qui permettront d'y accéder :

```
FIELD #1,2 AS CODE$,20 AS NOM$,4 AS PRIX$,2 AS NOMBRE$
```

Chaque fois que l'on voudra écrire des données dans le fichier, il faudra passer par les étapes suivantes :

(Nous supposerons, pour notre exemple, que C% est le code, N$ le nom de l'article, PX son prix et QTE% le nombre restant en stock).

**1.** Placer les données dans les variables réservées, en convertissant les données numériques en chaînes :

```
LSET CODE$=MKI$(C%)
LSET NOM$=N$
LSET PRIX$=MKS$(PX)
LSET NOMBRE$=MKI$(QTE%)
```

**2.** Écrire l'enregistrement dans le fichier :

```
PUT #1,1
```

Et inversement, quand on voudra relire les données de cet enregistrement numéro 1 :

**1.** Relire l'enregistrement sur disque :

```
GET #1,1
```

**2.** Récupérer les valeurs contenues dans les variables réservées, en reconvertissant en valeurs numériques les chaînes qui doivent l'être :

```
C%=CVI(CODE$)
N$=NOM$
PX=CVS(PRIX$)
QTE%=CVI(NOMBRE$)
```

Le programme suivant vous permettra de créer le fichier que nous venons de décrire, puis d'y inscrire automatiquement les données correspondant à 10 articles. Celles-ci peuvent ensuite être récupérées par leur numéro :

```
 5 '  CREATION FICHIER
 6 '
10 CLOSE:CLS
20 OPEN "A:ARTICLES.FCH" AS #1 LEN=28
30 FIELD #1,2 AS CODE$,20 AS NOM$,4 AS
   PRIX$, 2 AS NOMBRE$
34 '
35 '  ECRITURE DONNEES
36 '
40 FOR REC=1 TO 10
50 READ C%,N$,PX,QTE%
60 LSET CODE$=MKI$(C%):LSET NOM$=N$:LSET
   PRIX$=MKS$(PX): LSET NOMBRE$=
   MKI$(QTE%)
70 PUT #1,REC
80 NEXT REC:CLOSE
84 '
85 '  RELECTURE DES DONNEES
86 '
90 OPEN "A:ARTICLES.FCH" AS #1 LEN=28
100 FIELD #1,2 AS CODE$,20 AS NOM$,4 AS
    PRIX$, 2 AS NOMBRE$
110 INPUT "No ENREGISTREMENT ",REC
120 IF REC<0 OR REC>10 THEN 110
130 GET #1,REC
140 C%=CVI(CODE$):N$=NOM$:PX=CVS(PRIX$):
    QTE%=CVI(NOMBRE$)
150 PRINT "CODE:";C%;"  DESIGNATION:"
    ;N$;"  PRIX:";PX;"  RESTE:";QTE%
160 GOTO 110
164 '
165 '  DATAS
166 '
200 DATA 8,VIS 3MM,2.25,120,3,VIS
    4MM,3.10,655,1,CLOUS,0.45,1235
210 DATA 4,PORTE,542.50,3,2,FENETRE,
    212.22,8,5,GONDS,12.40,164
220 DATA 9,SERRURE,125.76,34,7,BUTOIR,
    3.47,56,10,PLAQUE,11.85,24
230 DATA 6,ENJOLIVEUR,23.10,16
```

# 5. Le graphique MSX

## 5.1. GÉNÉRALITÉS

En plus des deux modes texte qui ne permettent de dessiner qu'avec des caractères déjà formés, le MSX dispose de deux modes graphiques pour lesquels tous les points de l'écran sont accessibles individuellement par des instructions spécialisées :

— Le mode haute-résolution, 256 × 192 points.

— Le mode basse-résolution, 64 × 48 blocs. Chaque bloc est constitué de 4 / 4 points en haute-résolution.

On accède aux deux modes graphiques par l'instruction SCREEN, avec SCREEN 2 pour le mode haute-résolution et SCREEN 3 pour le mode

basse-résolution. La sélection du mode graphique n'est possible qu'à l'intérieur d'un programme : Le retour au mode direct implique automatiquement l'effacement de l'écran et la resélection du mode texte (SCREEN 0). Une fois l'un des deux modes choisi, toutes les instructions du graphique fonctionnent de manière identique, seules les coordonnées d'affichage sur l'écran sont interprétées différemment.

Les règles d'apparition de la couleur sont légèrement différentes du mode texte : L'option /Couleur avant-plan/ de l'instruction COLOR ne modifie pas instantanément le contenu de l'écran, mais uniquement la couleur d'écriture pour les instructions du graphique. Il est donc théoriquement possible de définir séparément la couleur de chaque point de l'écran en mode graphique ! En réalité, ce n'est vrai qu'en mode basse-résolution : Certaines limites de conception interdisent la coexistence à l'intérieur d'un même octet de plus de deux couleurs en haute-résolution. En pratique, cette limitation s'avère peu gênante, on peut ne pas en tenir compte lors de la création d'un dessin à l'écran.

Un certain nombre d'instructions est spécifique au mode graphique, dont la puissante instruction DRAW qui combine le dessin en coordonnées relatives et en coordonnées absolues. Il faut ajouter à ces commandes la possibilité de gérer des "sprites", ces petits caractères dont on peut définir la taille et le contenu et insérer dans les graphismes. Comme dans le chapitre consacré au Basic d'usage général, ces instructions seront détaillées et parfois associées à des programmes d'exemples.

Le mélange texte et graphique est possible en adressant l'écran graphique par PRINT#, après l'avoir ouvert comme un fichier par OPEN. La procédure sera détaillée au paragraphe 5.2, après la présentation des instructions du graphique.

## 5.2. LES INSTRUCTIONS DU GRAPHIQUE

Toutes les commandes du graphique font référence à la position d'un curseur à l'écran. Ce curseur est différent de celui qu'emploie le mode texte : Il n'apparaît jamais physiquement, sa position est gérée directement en mémoire.

— La coordonnée horizontale (ou abscisse) du curseur doit être comprise entre 0 et 255. Dans la syntaxe des commandes, cette valeur est représentée par X.

— Sa coordonnée verticale (ou ordonnée) entre 0 et 191. Il s'agit de Y dans la syntaxe.

— Le point de coordonnées 0,0 ou point d'origine, correspond au coin supérieur gauche de l'écran.

REMARQUES :

— On appelle "couleur en vigueur" les couleurs d'affichage sélectionnées par COLOR pour l'Avant-plan, selon les mêmes règles de syntaxe que celles définies au paragraphe 4.1.1 pour l'affichage texte.

— En mode basse-résolution, toutes les instructions du graphique effectuent une approximation sur les coordonnées pour définir quelles sont les correspondances entre les points d'affichage pour les deux modes. Un bloc en basse-résolution étant formé de 4 × 4 points élémentaires, on peut en déduire :

Tous les points dont les abscisses sont comprises entre 0+X et 3+X en haute-résolution correspondent à l'abscisse 0+X en basse-résolution.

Tous les points dont les ordonnées sont comprises entre 0+Y et 3+Y en haute-résolution correspondent à l'ordonnée 0+Y en basse-résolution.

X doit être un entier divisible par 4 compris entre 0 et 250.

Y doit être un entier divisible par 4 compris entre 0 et 188.

### 5.2.1. Instructions et fonctions d'usage général

#### CIRCLE

*Syntaxe* : CIRCLE [STEP] (X,Y),/Rayon/[,/Couleur/] [,/Angle début/] [,/Angle fin/] [,/Aspect/]

*Application* : Trace un cercle, un arc ou une ellipse dont le centre se place au point de coordonnées X,Y et dont le rayon est défini par /Rayon/. Les nombreuses options disponibles permettent de modifier l'aspect, la couleur et la taille du cercle :

— Si STEP est omis, le centre du cercle se place au point de coordonnées absolues X,Y avec X pour abscisse et Y pour ordonnée. Dans le cas contraire, X et Y servent à définir un déplacement sur l'échelle des coordonnées, relativement à la position courante du curseur graphique.

| | |
|---|---|
| `CIRCLE (50,60),30` | Trace un cercle de rayon 30 au point de coordonnées 50,60. |
| `CIRCLE STEP(10,0),30` | Trace un cercle de rayon 30 dont le centre se place 10 points à droite de la position courante du curseur graphique. |
| `CIRCLE STEP(0,-10),50` | Trace un cercle de rayon 50, dont le centre se place 10 points au-dessus du curseur graphique. |

— /Couleur/ sert à définir la couleur du tracé. Si elle est omise, la couleur de l'avant plan sélectionnée dans COLOR est prise par défaut.

| | |
|---|---|
| `CIRCLE (60,50),50,6` | Trace un cercle rouge de rayon 50. |

— /Angle début/ et /Angle fin/ permettent le dessin d'une portion de cercle ou d'ellipse en définissant sur quel point de la circonférence le tracé doit démarrer et s'arrêter. Ces deux paramètres sont exprimés en radians, et doivent être compris entre 0 et 2*PI (PI=3.14159). Si l'on assimile un cercle complet au cadran d'une montre, la valeur 0 correspond à 3 heures, PI à 9 heure et 2*PI à un tour complet.

| | |
|---|---|
| `CIRCLE (40,50),50,6,PI/2,PI` | Trace un quart de cercle rouge au point de coordonnées 40,50 démarrant à 12 heures et de rayon 50. |

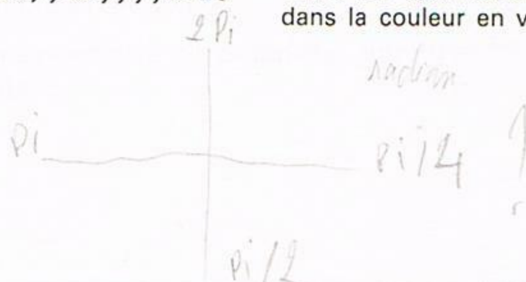
— /Aspect/ fixe le rapport entre le rayon horizontal et le rayon vertical d'une ellipse. La valeur choisie par défaut dépend de l'ordinateur, mais permet de tracer directement des cercles sans spécifier de rapport.

| | |
|---|---|
| `CIRCLE (50,60),50,6,0,PI,0.25` | Trace une demi-ellipse dont le rayon horizontal vaut à peu près quatre fois le rayon vertical. |

REMARQUES :

— Il est possible d'omettre l'une ou plusieurs des options de la liste (les virgules servent de séparateurs) :

| | |
|---|---|
| `CIRCLE (50,60),50,,PI,2*PI` | Trace une ellipse complète de rayon 50 au point de coordonnées 50,60 dans la couleur en vigueur. |
| `CIRCLE (50,60),50,,,0.25` | Trace un demi-cercle de rayon 50 dans la couleur en vigueur. |



— Si /Angle début/ ou /Angle fin/ est négatif, il est converti en positif mais un trait vient relier le centre du cercle ou de l'ellipse à sa périphérie.

— X et Y doivent être des entiers positifs ou négatifs. Si les valeurs sont supérieures ou inférieures aux bords externes de l'écran, elles sont automatiquement remplacées par la coordonnée autorisée la plus proche.

***Exemple*** :

```
10 ' DEMO CERCLES
20 '
30 COLOR 15,1,1:SCREEN 2:C=2
40 FOR X=0 TO 255 STEP 16
50 Y1=SIN(4*3.1416*X/255)*30+50
60 Y2=COS(4*3.1416*X/255)*30+140
70 CIRCLE (X,Y1),7,C
80 CIRCLE (X,Y2),7,C+1
90 C=C+1:IF C>14 THEN C=2
100 NEXT
110 X$=INPUT$(1)
```

**Explications :**

— 30 : Sélection de la couleur et de l'écran haute-résolution.

— 40 : Boucle faisant varier X linéairement.

— 50-60 : Variation de Y1 et Y2 en fonction de X, selon une courbe sinusoïdale.

— 70-80 : Affichage aux coordonnées X,Y1 et Y,Y2 de deux cercles de rayon 7 dans deux couleurs différentes dont les numéros se suivent.

— 90 : Incrément de la couleur. Si la valeur obtenue est hors des limites admises par le MSX-BASIC, la variable C est réinitialisée.

— 110 : Une fois la boucle terminée, cette ligne interdit le retour au mode direct (et donc l'effacement de l'écran) tant qu'une touche n'est pas tapée au clavier.

## LINE

*Syntaxe* : LINE [[STEP] (X1,Y1)]–[STEP] (X2,Y2) [,/Couleur/] [,B ou BF]

*Application* : Trace une ligne entre les points de coordonnées X1,Y1 et X2,Y2 :

— Si STEP est omis, la droite est tracée entre les points de coordonnées absolues X1,Y1 et X2,Y2 avec X pour les abscisses et Y pour les ordonnées. Dans le cas contraire, X et Y servent à définir un

déplacement sur l'échelle des coordonnées, relativement à la position courante du curseur graphique.

| | |
|---|---|
| `LINE (0,0)-(100,100)` | La ligne est tracée entre les points de coordonnées absolues (0,0) et (100,100). |
| `LINE STEP(10,0)-(100,100)` | Le début du tracé démarre 10 points à droite de la position courante du curseur graphique. |
| `LINE (0,0)-STEP(30,0)` | La ligne est tracée entre le point de coordonnées (0,0) et un point situé 30 points à droite de la position du curseur graphique. |
| `LINE STEP(10,0)-STEP(60,-10)` | Cette fois, les deux points sont référencés par rapport à la position courante du curseur. |

— Le couple de coordonnées X1,Y1 peut être omis. Dans ce cas, la ligne est tracée entre la position courante du curseur et X2,Y2.

| | |
|---|---|
| `LINE -(100,100)` | La ligne est tracée entre la position du curseur et le point de coordonnée 100,100. |
| `LINE -STEP(10,0)` | La ligne est tracée entre la position courante du curseur et un point décalé de 10 points vers la droite sur l'axe des abscisses. |

— /Couleur/ sert à définir la couleur du tracé. Si elle est omise, la couleur de l'avant plan sélectionnée dans COLOR est prise par défaut.

| | |
|---|---|
| `LINE (0,0)-(100,100),6` | Trace une ligne en rouge. |

— Si l'option B est inclue dans la syntaxe, LINE trace un rectangle dont la diagonale est figurée par la ligne à tracer. BF a un effet similaire, mais remplit l'intérieur du rectangle dans la couleur en vigueur.

| | |
|---|---|
| `LINE (0,0)-(100,100),6,B` | Trace un rectangle rouge dont les coins supérieur gauche et inférieur droit ont pour coordonnées (0,0) et (100,100). |

REMARQUES:

— Il est possible d'omettre l'une ou plusieurs des options de la liste:

```
LINE -(100,100),,B
```

Trace un rectangle dans la couleur en vigueur dont le coin supérieur droit se place à la position du curseur graphique.

— X et Y doivent être des entiers positifs ou négatifs. Si les valeurs sont supérieures ou inférieures aux bords externes de l'écran (255 et 191), elles sont automatiquement remplacées par la coordonnée autorisée la plus proche.

— Après exécution de LINE, la position courante du curseur graphique est déplacée au point d'arrivée de la droite (X2,Y2).

***Exemple*** :

```
 10 ' DEMO BOITES EN COULEUR
 20 '
 30 SCREEN 2:COLOR 4,1:CLS
 40 LINE (0,0)-(255,191),,B
 50 FOR C=1 TO 15
 60 A1=INT(RND*255):B1=INT(RND*191)
 70 A2=INT(RND*255):B2=INT(RND*191)
 80 IF A1<3 OR B1<3 OR A2<3 OR B2<3 THEN 60
 90 IF ABS(A2-A1)>100 OR ABS(B2-B1)>100 OR ABS(A2-A1)<40 OR
    ABS(B2-B1)<40 THEN 60
110 LINE(A1,B1)-(A2,B2),C,BF
120 NEXT
130 X$=INPUT$(1)
```

**Explications :**

— 30 : Sélection de la couleur, passage en haute-résolution.

— 40 : Tracé d'un cadre dont la taille correspond aux bords externes de l'écran.

— 50 : Boucle faisant varier la couleur C entre 1 et 15.

— 60-70 : Attribution de valeurs aléatoires comprises entre 0 et 255 pour les abscisses A1 et A2, et 0 et 191 pour les ordonnées B1 et B2 respectivement.

— 80-90 : Choix de valeurs limites pour l'emplacement et la taille des rectangles.

— 110 : Tracé de boîtes aux points de coordonnées A et B, dans la couleur C.

— 130 : Blocage en mode programme jusqu'à l'appui sur une touche du clavier.

## PSET/PRESET

*Syntaxe*: PRESET [STEP] (X,Y) [,/Couleur/]
PSET [STEP] (X,Y) [,/Couleur/]

*Application*: Si /Couleur/ est omise, PSET affiche un point élémentaire en X,Y dans la couleur en vigueur et PRESET l'efface (le point est affiché dans la couleur choisie pour le fond dans COLOR).

— Si STEP est omis, le point s'affiche aux coordonnées absolues X,Y avec X pour les abscisses et Y pour les ordonnées. Dans le cas contraire, X et Y servent à définir un déplacement sur l'échelle des coordonnées, relativement à la position courante du curseur graphique.

`PSET (100,100)` Affiche un point en 100,100.

`PRESET STEP(0,10)` Efface un point situé 10 crans en dessous du curseur graphique.

— L'emploi de /Couleur/ rend les deux instructions identiques: Un point est affiché dans la couleur choisie aux coordonnées X,Y.

REMARQUE: Après exécution de PSET ou PRESET, le curseur graphique est déplacé au point X,Y, qui devient la nouvelle position courante.

***Exemples***: Programmes de dessin par les flèches du clavier.

```
10 ' DESSIN DANS QUATRE DIRECTIONS
20 '
30 COLOR 15,1:SCREEN 2
40 X=100:Y=100
50 PSET (X,Y),15
60 X$=INKEY$:IF X$="" THEN 60
80 IF X$=CHR$(28) AND X<255 THEN X=X+1
90 IF X$=CHR$(29) AND X>0 THEN X=X-1
100 IF X$=CHR$(30) AND Y>0 THEN Y=Y-1
110 IF X$=CHR$(31) AND Y<191 THEN Y=Y+1
120 GOTO 50
```

**Explications :**

— 40: Fixe le point de départ pour le dessin.

— 50: Affichage d'un point par PSET.

— 60: Saisie d'une touche du clavier.

— 80-110: Si le caractère saisi correspond à un des quatre codes flèches, les coordonnés du curseur sont incrémentées en fonction du sens choisi (voir liste des codes de contrôle au § 1.4.3).

— 120: Retour à l'affichage.

La fonction STICK(N) lit le clavier si N=0. Elle autorise une saisie des huit directions :

```
10 ' DESSIN DANS HUIT DIRECTIONS
20 '
30 COLOR 15,1:SCREEN 2
40 X=100:Y=100
50 PSET (X,Y),15
60 A=STICK(0):IF A=0 THEN 60
80 IF A=3 AND X<255 THEN X=X+1
90 IF A=7 AND X>0 THEN X=X-1
100 IF A=1 AND Y>0 THEN Y=Y-1
110 IF A=5 AND Y<191 THEN Y=Y+1
120 IF A=8 AND X>0 AND Y>0 THEN X=X-1:Y=Y-1
130 IF A=6 AND X>0 AND Y<191 THEN X=X-1:Y=Y+1
140 IF A=2 AND X<255 AND Y>0 THEN X=X+1:Y=Y-1
150 IF A=4 AND X<255 AND Y<191 THEN X=X+1:Y=Y+1
200 GOTO 50
```

**Explications :**

Le programme utilise la valeur retournée par la fonction STICK au lieu des codes de contrôle pour gérer les déplacements du curseur. Si STICK(0)=0, aucune touche n'est pressée.

*Voir aussi* : POINT.

***Exemple*** :

```
10 ' DEMO POINTS BASSE RESOLUTION
20 '
30 SCREEN 3
40 X=RND(1)*255
50 Y=RND(1)*191
60 C=RND(1)*15
70 PSET (X,Y),C
80 GOTO 40
```

aléatoire — Screen 7 — *511 — *211 — *15

Ce petit programme remplit l'écran avec des blocs basse-résolution, à un emplacement et dans une couleur aléatoire. Pour paser en haute-résolution, il suffit de remplacer la ligne 30 par SCREEN 2.

**POINT**

*Syntaxe* : POINT (X,Y)

*Application* : La fonction POINT retourne la couleur du point de coordonnées X,Y.

*Voir aussi* : PSET, PRESET.

exemple :

IF Point(X,Y)=15 then ....

si le point de coordonné X,Y est de couleur blanc. Alors....

## PAINT

*Syntaxe*: PAINT [STEP] (X,Y) [,/Couleur/] [,/Bordure/]

*Application*: Remplit une zone de l'écran ou une figure géométrique dans la couleur choisie: X et Y sont les coordonnées d'un point de la zone à remplir.

— Si STEP est omis, le point de départ du remplissage se place aux coordonnées absolues X,Y avec X pour les abscisses et Y pour les ordonnées. Dans le cas contraire, X et Y servent à définir un déplacement sur l'échelle des coordonnées, relativement à la position courante du curseur graphique.

— /Couleur/ définit la couleur dans laquelle la zone sera "noircie". Elle prend par défaut la valeur en vigueur pour l'avant plan.

— /Bordure/ sert à définir la couleur du pourtour à colorer en basse-résolution. En mode haute-résolution, /Bordure/ prend automatiquement la même valeur que /Couleur/, et peut donc être omise.

REMARQUE: X et Y ne doivent pas dépaser les valeurs limites de l'écran haute-résolution (255 et 191).

***Exemples***:

Le remplissage d'une boîte par LINE avec l'option BF peut être remplacé par PAINT:

```
10 COLOR 15,5,1:SCREEN 2
20 LINE (0,0)-(100,150),6,BF
30 X$=INPUT$(1)
```

est équivalent à:

```
10 COLOR 15,5,1:SCREEN 2
20 LINE (0,0)-(100,150),6,B
30 PAINT(50,50),6
40 X$=INPUT$(1)
```

PAINT peut servir à remplir des formes géométriques complexes:

```
10 COLOR 15,5,1:SCREEN 2
20 CIRCLE (255,100),50,10
30 PAINT  (255,100),10,10
40 X$=INPUT$(1)
```

### 5.2.2. L'instruction DRAW

Plus qu'une instruction, DRAW constitue un macro-langage graphique: le dessin dépend d'une suite de commandes inclue à l'intérieur d'une chaîne de caractères. La possibilité d'introduire des variables dans la chaîne et de modifier le facteur d'échelle d'un dessin ouvre la voie vers le zooming et la mémorisation d'un catalogue de formes géométriques!

*Syntaxe*: DRAW /Chaîne de caractères/

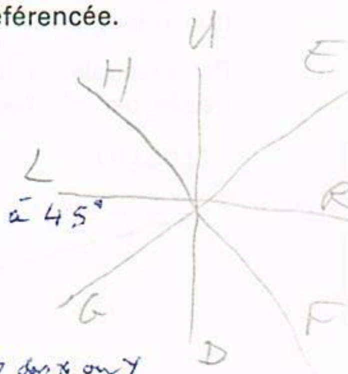
*Application*: On utilise DRAW pour dessiner à l'aide de commandes à une lettre qui font partie de la chaîne spécifiée en argument. Cette chaîne définit un objet qui est dessiné lorsque le Basic exécute l'instruction DRAW. /Chaîne de caractères/ doit être une constante entourée de guillemets, et sa longueur ne doit pas dépasser 255 caractères.

**1. Les commandes de mouvement:**

— Les commandes de mouvement suivantes induisent un déplacement du curseur graphique **avec traçage** d'une ligne à partir de sa dernière position référencée: Il s'agit donc d'un déplacement relatif.

Après chaque commande, le curseur est placé sur le dernier point dessiné qui devient alors la nouvelle position référencée.

**U** n Déplacement vers le haut.
**D** n Déplacement vers le bas.
**L** n Déplacement vers la gauche.
**R** n Déplacement vers la droite.
**E** n Déplacement en diagonale vers le haut à droite.
**F** n Déplacement en diagonale vers le bas à droite.
**G** n Déplacement en diagonale vers le bas à gauche.
**H** n Déplacement en diagonale vers le haut à gauche.




Pour chacune de ces commandes, n représente la distance de déplacement. La longueur en nombre de points est de n fois le facteur d'échelle du dessin, spécifié par la commande S (voir plus loin). Si S est omise, n représente directement la longueur du déplacement (S=4).

***Exemples***:

```
10 COLOR 15,5,1:SCREEN 2
20 LINE (0,0)-(100,100)
30 DRAW "R50U50L50D50"
40 X$=INPUT$(1)
```

Trace un carré de 50 de côté à partir du point de coordonnées (100,100). On peut constater qu'il n'est pas nécessaire de séparer chaque commande de la suivante par un point-virgule.

```
10 COLOR 15,5,1:SCREEN 2
20 LINE (0,0)-(100,100)
30 DRAW "E15F15L30"
40 X$=INPUT$(1)
```

Trace un triangle dont la base fait 30 points de long.

— Il est également possible de déplacer le curseur vers un point précis de l'écran:

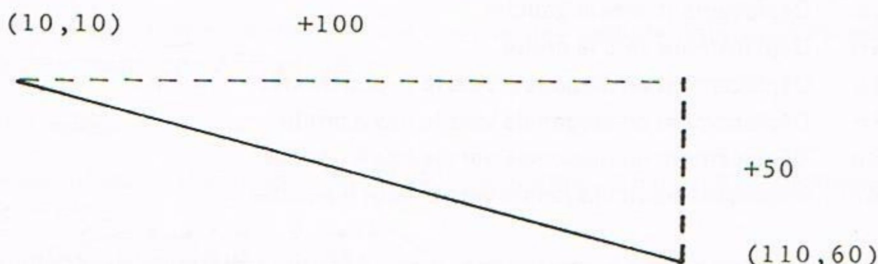
**M** X,Y Déplacement du curseur vers le point de coordonnées X,Y avec X pour abscisse et Y pour ordonnée. X et Y doivent être compris dans les valeurs limites autorisées (255 et 191).

Le programme de carré précédent peut s'écrire :

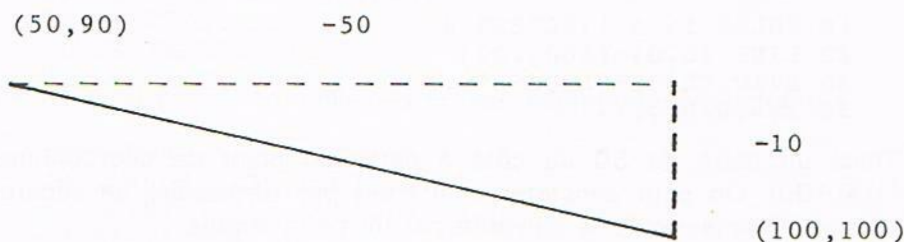
```
10 COLOR 15,5,1:SCREEN 2
20 DRAW "M100,100R50U50L50D50"
40 X$=INPUT$(1)
```

La commande **M** peut servir dans les déplacements relatifs, en faisant précéder les arguments X et Y du signe + ou –. Dans ce cas, le curseur est déplacé du nombre de points fixé par X et Y, et non au point de coordonnées X,Y. Comme pour les commandes de mouvement relatif U,D,L,R,E,F,G,H, ce mouvement s'effectue à partir de la dernière position référencée.

Par exemple, si la dernière position référencée est (10,10), la droite tracée par M+100,+50 peut se schématiser ainsi :




Les valeurs négatives sont acceptées, elles entraînent un déplacement inversé sur l'échelle des abscisses et des ordonnées. Soit (100,100) la position du curseur, M–50,–10 aura pour effet de tracer une droite vers le point de coordonnées (50,90) :




Deux préfixes de commande peuvent précéder n'importe laquelle des commandes de mouvement ci-dessus :

**B** Le curseur est déplacé sans traçage de points.

**N** Après déplacement avec ou sans traçage, le curseur retourne à la dernière position référencée.

***Exemples*** :

```
10 ' DESSIN DE 2 CARRES TETES-BECHES
15 '
20 COLOR 15,5,1:SCREEN 2
30 DRAW "BM100,100":'CURSEUR AU CENTRE DE L'ECRAN
40 GOSUB 100
50 DRAW "BL50BU50":' CURSEUR A GAUCHE DU PREMIER CARRE
60 GOSUB 100
70 X$=INPUT$(1)
80 END
100 DRAW "U50R50D50L50":' DESSIN DU CARRE
110 RETURN
```

```
10 ' DESSIN DE RAYONS
20 '
30 COLOR 15,5,1:SCREEN 2
40 DRAW "BM100,100"
50 DRAW "NU50NE50NR50NF50ND50NG50NL50NH50"
60 X$=INPUT$(1)
```

## 2. Les commandes d'usage général :

Quatre autres commandes viennent s'ajouter aux fonctions de déplacement du curseur :

**A** n — Établit l'angle n sous lequel le dessin est affiché. n peut être compris entre 0 et 3. Si n=0, le dessin est affiché normalement, c'est le mode par défaut. Si n=1, le dessin est pivoté de 90° dans le sens inverse des aiguilles d'une montre, si n=2 de 180° et si n=3 de 270°.

**C** n — Établit la couleur du tracé n, avec n compris entre 0 et 15. La valeur par défaut est la couleur de l'avant-plan sélectionnée dans COLOR.

**S** n — Établit le facteur d'échelle, avec n compris entre 1 et 255. Pour obtenir le facteur d'échelle, il suffit de diviser n par 4. Par exemple, si n=1 le facteur est 1/4. Ce facteur multiplié par les distances données en argument des commandes de mouvement relatif U,D,L,R,E,F,G,H et M donne la distance réelle de déplacement. La valeur par défaut de 4 donne un facteur d'échelle de 1.

taille normal n=4
donc S4

**X** A$ ; La commande X permet d'exécuter une sous-chaîne à l'intérieur de la constante chaîne donnée en argument de DRAW. Le point-virgule en fin d'expression est obligatoire.

***Exemples*** :

Les exemples qui suivent sont tous valides dans les deux modes, haute et basse résolution : Seul l'aspect du dessin est modifié.

```
10 ' DESSIN D'UNE LETTRE E CONTENUE DANS A$
20 '
30 CLS:SCREEN 3
40 A$="BM100,120C6R40U10L30U20R10U10L10U20R30U10L40D70"
50 DRAW "XA$;"
60 X$=INPUT$(1)
```

Il est possible de dessiner plusieurs chaînes successivement avec une seule instruction DRAW, ce qui permet le dessin d'une chaîne de longueur non limitée à 255 caractères, ou encore le stockage de formes géométriques dans des sous-chaînes :

```
10 ' DESSIN DE DEUX LETTRES E CONTENUES DANS A$,B$
20 '
30 CLS:SCREEN 3
40 A$="BM100,120C6R40U10L30U20R10U10L10U20R30U10L40D70"
45 B$="BM180,120C5R40U10L30U20R10U10L10U20R30U10L40D70"
50 DRAW "XA$;XB$;"
60 X$=INPUT$(1)
```

Les commandes **A** et **S** permettent de modifier l'orientation et la taille du dessin :

```
10 ' DESSIN DE DEUX LETTRES E CONTENUES DANS A$,B$
20 '
30 CLS:SCREEN 3
40 A$="A0S2BM100,120C6R40U10L30U20R10U10L10U20R30U10L40D70"
45 B$="A2S4BM180,120C5R40U10L30U20R10U10L10U20R30U10L40D70"
50 DRAW "XA$;XB$;"
60 X$=INPUT$(1)
```

Ce programme peut également s'écrire :

```
10 ' DESSIN DE DEUX LETTRES E CONTENUES DANS A$
20 '
30 CLS:SCREEN 3
40 A$="R40U10L30U20R10U10L10U20R30U10L40D70"
50 DRAW "BM100,120C6A0S2XA$;BM180,120C5A2S4XA$;"
60 X$=INPUT$(1)
```

REMARQUE : Dans toutes les commandes, l'argument n, x ou y peut être une constante numérique ou l'expression =/**Variable**/**;** dans laquelle /Variable/ représente le nom d'une variable numérique. Comme pour **X**

le point-virgule (;) est obligatoire en fin d'expression lorsqu'on utilise une variable de cette façon.

***Exemples*** : Les espaces ne sont pas significatifs dans DRAW, les commandes peuvent toutes être groupées les unes après les autres :

```
10 ' TRACE D'UN CARRE A L'AIDE DE VARIABLES
15 SCREEN 2:CLS
20 X=120:Y=100:COTE=50
30 DRAW "BM=X;,=Y;L=COTE;U=COTE;R=COTE;D=COTE;"
40 X$=INPUT$(1)
```

La possibilité d'inclure des variables dans une chaîne autorise la répétition d'un même dessin, en modifiant ou non son orientation :

```
10 ' CHANGEMENT D'ORIENTATION
20 '
30 SCREEN 3:CLS
40 FOR I%=0 TO 3 STEP 2
50 A$="A=I%;BM120,95C6R40U10L30U20R10U10L10U20R30U10L40D70"
60 DRAW "XA$;"
70 NEXT
80 X$=INPUT$(1)
```

Le dessin en relatif autorise des effets de "zooming", en paramétrant l'échelle et la position du graphique sur l'écran :

```
10 ' EFFET ZOOM
20 '
30 SCREEN 3:CLS
40 PSET (5,180),6
50 FOR I%=1 TO 9 STEP 3
70 DRAW "A0C6S=I%;BM+30,-10R40U10L30U20R10U10L10U20R30U10L40D70"
80 NEXT
90 X$=INPUT$(1)
```

*Note* : Lorsqu'on fournit à DRAW une coordonnée en dehors des limites autorisées, celle-ci est automatiquement remplacée par la valeur valide la plus proche. En d'autres termes, les valeurs négatives deviennent 0, les valeurs X supérieures à 255 deviennent 255, et les valeurs Y supérieures à 191 deviennent 191.

### 5.2.3. Un programme d'exemple

Le programme suivant récapitule les instructions et fonctions spécialisées du graphique. Il s'agit d'un programme de base sur lequel viendront se greffer des fonctions supplémentaires dans les paragraphes consacrés à la gestion des sprites et au mélange texte-graphique :

BM 100,100

ou BM=X;,=Y;



```
10 ' INITIALISATIONS
20 '
30 KEY OFF
40 COLOR 15,5,1
50 SCREEN 2,0
60 '
70 ' DESSIN DU CIEL
80 '
90 FOR N%=1 TO 200
100 X=RND(1)*255:Y=RND(1)*191
110 PSET(X,Y),11
120 NEXT
130 '
140 ' DESSIN DES PLANETES
150 '
160 R=3.141593#/180
170 CIRCLE (255,10),50,10
180 PAINT (255,10),10,10
190 CIRCLE (255,10),70,14,R*130,,.25
200 CIRCLE (255,10),80,14,R*100,,.35
210 PAINT (255,35),14,14
220 CIRCLE (40,30),20,9
230 PAINT (40,30),9,9
240 CIRCLE (120,100),20,11,R*100,R*265,1
250 CIRCLE (140,100),30,11,R*140,R*225,1
260 PAINT (105,100),11,11
270 CIRCLE (20,160),20,8,,,1
280 PAINT (18,150),8,8
290 CIRCLE (40,160),8,8,,,1
300 CIRCLE (155,370),200,3,R*50,R*130,1
310 PAINT (155,185),3,3
320 '
330 X$=INPUT$(1)
```

c'est mieux

pi=3.141593 : R=Pi/180




circle (X,Y),R,C,A1,A2,R

Rayon, couleur, Angle depart, Angle fin, ...

**Explications :**

— 30 : Supprime l'affichage des touches de fonction sur la vingt quatrième ligne.

— 40-50 : Sélection de la couleur et de la haute-résolution.

— 70-120 : Affichage d'étoiles jaune dans le ciel, à des coordonnées X et Y aléatoires.

— 140-310 : Dessin de planètes avec l'instruction CIRCLE. La valeur choisie pour R (ligne 160) permet de spécifier directement en degrés les portions d'arcs à afficher. PAINT sert à remplir dans les couleurs choisies les diverses formes créées (Mars, Saturne, etc...).

### 5.2.4. Le mélange texte-graphique

Pour afficher du texte sur un écran graphique, il suffit d'ouvrir par OPEN un fichier en mode OUTPUT vers ce dernier, puis d'écrire directement par PRINT# sur l'écran : Le texte s'affiche à la position courante du curseur graphique, dans la couleur en vigueur.

On peut visualiser cette possibilité en ajoutant au programme qui précède :

```
330 ' AFFICHAGE DU TEXTE
340 '
350 OPEN "GRP:" FOR OUTPUT AS #1
360 DRAW "BM10,0"
370 PRINT #1,"L'UNIVERS DU MSX"
380 CLOSE
390 '
400 X$=INPUT$(1)
```

## 5.3. LA GESTION DES "SPRITES"

De nombreux logiciels de jeux gèrent des processus d'animation sur des motifs et des personnages devenus classiques : Invaders, soucoupes volantes, notes de musiques. Pour faciliter la mise au point de ce type de programmes, le MSX-BASIC dispose d'un groupe de quatre instructions destinées à créer et à contrôler des motifs graphiques programmables appelés "sprites" (lutins en français). On peut par l'intermédiaire de ces commandes définir point par point des motifs de taille et d'orientation variables, qu'il est ensuite possible d'afficher, de déplacer et de faire disparaître sans modifier le dessin déjà présent sur l'écran.

## 5.3.1. Le principe

Pour bien utiliser les instructions de contrôle des sprites, il faut comprendre le principe de fonctionnement de ces caractères :

Un sprite est un caractère inclu dans une matrice de 8 × 8, chacun des points de la matrice pouvant être "allumé" (noir) ou "éteint" (blanc). Un point est représenté par un bit : Il faudra donc un octet pour représenter 8 points ou une ligne horizontale de la matrice, et 8 octets pour stocker un caractère complet dans l'instruction dc création SPRITE$ (voir § 5.3.2). Une fois le sprite créé, un point de la matrice correspond lors de l'affichage à un point sur l'écran en mode haute-résolution graphique. L'exemple suivant montre comment un "invader" peut être dessiné selon cette méthode :

| *Matrice* | *Valeur décimale* | *Équivalent binaire* |
|---|---|---|
| | 12 | 00001100 |
| | 30 | 00011110 |
| | 45 | 00101101 |
| | 63 | 00111111 |
| | 30 | 00011110 |
| | 12 | 00001100 |
| | 30 | 00011110 |
| | 51 | 00110011 |

Chaque ligne horizontale de la matrice peut être représentée par son équivalent binaire, ou par la valeur décimale correspondante.

Cette dernière se calcule selon la méthode décrite au paragraphe 3.2, dans la partie consacrée au stockage des données :

Soit le nombre binaire N, avec N=ABCEFGH, A,B,C,D,E,F,G,H ne pouvant prendre que la valeur 0 ou 1. Si l'on considère que le bit de gauche est le plus significatif, la valeur décimale D est égale à :

```
D=A*2^7+B*2^6+C*2^5+D*2^4+E*2^3+F*2^2+G*2^1+H*2^0   ou enco
```

```
D=A*128+B*64+C*32+D*16+E*8+F*4+G*2+H
```

ou encore :

— Pour dessiner un sprite de taille standard, il suffit de suivre une méthode simple :

1) Dessinez sur le papier un damier de 8 colonnes et 8 lignes.

2) Noircissez les cases appropriées.

3) A droite, à côté de chaque ligne, notez le nombre décimal ou binaire résultant pour chaque ligne de la matrice.

4) Saisissez les valeurs trouvées à l'aide de SPRITE$ (voir plus loin).

REMARQUES :

— L'instruction SCREEN permet la sélection de quatre tailles différentes pour les sprites (voir SCREEN, SPRITE$) :

8 × 8, taille normale ou double taille.

16 × 16, taille normale ou double taille.

En fait, un sprite de 16 × 16 cases est constitué de 4 éléments de taille standard 8 × 8 placés côte à côte. Le format "double taille" ne modifie que l'aspect, pas le nombre de points servant à les définir.

— Une fois le sprite dessiné, l'instruction PUT SPRITE permet de le placer n'importe où à l'écran, de le faire disparaître, etc...

### 5.3.2. Les instructions

#### SPRITE$

*Syntaxe* : SPRITE$(N)=/Chaîne de caractères/

*Application* : Saisie dans une chaîne d'une suite d'octets correspondant au dessin d'un sprite. Suivant la taille des sprites choisie dans l'instruction SCREEN, le nombre N de sprites différents pouvant être défini varie :

Si la taille sélectionnée est 8 × 8 (SCREEN 2,0 ou SCREEN 2,1), N doit être compris entre 0 et 256.

Si la taille est 16 × 16 (SCREEN 2,2 ou SCREEN 2,3), chaque sprite fait quatre fois la taille standard, et N doit être compris entre 0 et 64.

— Pour mémoriser le dessin d'un sprite 8 × 8, on concatène dans une chaîne les huit caractères dont le code décimal est déduit du dessin des huit lignes horizontales de la matrice, puis on saisit cette chaîne dans SPRITE$ (en commençant par la ligne du haut). L'"invader" du paragraphe précédent peut s'écrire :

```
10 A$=CHR$(12)+ CHR$(30)+ CHR$(45)+ CHR$(63)+ CHR$(30)+ CHR$(12)+
     CHR$(30)+ CHR$(51)
20 SPRITE$(1)=A$
```

Pour simplifier la saisie des codes, le préfixe "&B" autorise l'entrée directe du contenu binaire de la matrice :

```
10 A$ = CHR$(&B00001100)+ CHR$(&B00011110)+ CHR$(&B00101101)+
        CHR$(&B00111111)+ CHR$(&B00011110)+ CHR$(&B00001100)+
        CHR$(&B00011110)+ CHR$(&B00110011)
20 SPRITE$(1)=A$
```

En insérant ce contenu binaire dans des lignes de DATA, il est possible de "dessiner" directement dans le programme le caractère à saisir, à l'aide de 0 et de 1 :

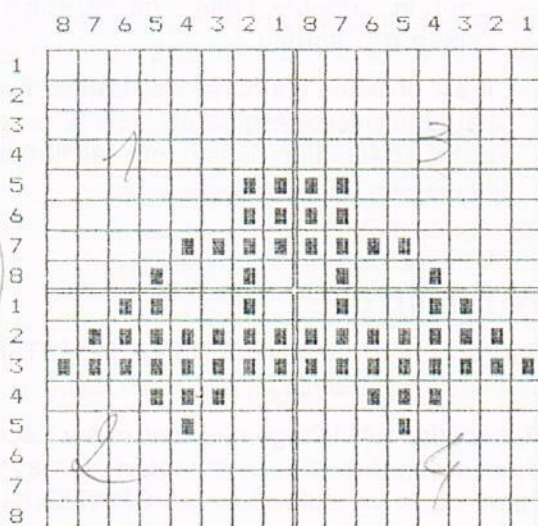
```
10 SCREEN 2,0:' INITIALISE EN 8 X 8 CASES
15 A$=""
20 FOR I=1 TO 8
20 READ X$
30 A$=A$+CHR$(VAL("&B"+X$))
40 NEXT I
50 SPRITE$(1)=X$
60 END
100 DATA 00001100
110 DATA 00011110
120 DATA 00101101
130 DATA 00111111
140 DATA 00011110
150 DATA 00001100
160 DATA 00011110
170 DATA 00110011
```

| | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | ■ | ■ | | |
| 2 | | | | ■ | ■ | ■ | ■ | |
| 3 | | | ■ | | ■ | ■ | | ■ |
| 4 | | | ■ | ■ | ■ | ■ | ■ | ■ |
| 5 | | | | ■ | ■ | ■ | ■ | |
| 6 | | | | | ■ | ■ | | |
| 7 | | | | ■ | ■ | ■ | ■ | |
| 8 | | | ■ | ■ | | | ■ | ■ |

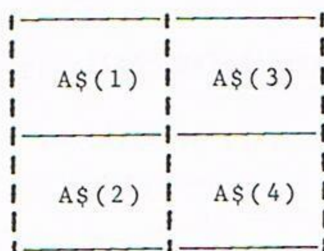
Cette méthode, de loin la plus simple à mettre en œuvre, est employée dans tous les programmes d'exemple du paragraphe 5.3.3. L'expression de la ligne 30 permet la conversion des chaînes lues dans les DATA en un code ASCII décimal équivalent.

— Un sprite 16 × 16 étant constitué de 4 sprites 8 × 8 placés côte à côte, on les définit de manière identique. Pour mémoriser le dessin d'un sprite en 16 × 16, on constitue 4 chaînes de 8 caractères comme précédemment, dont on saisit la somme dans SPRITE$ :

```
10 SCREEN 2,2:' INITIALISE EN 16 X 16 CASES
20 DIM A$(4)
40 FOR M%=1 TO 4
50 FOR N%=1 TO 8
60 READ X$
70 A$(M%)=A$(M%)+CHR$(VAL("&B"+X$))
80 NEXT M%
90 NEXT N%
100 SPRITE$(1)=A$(1)+A$(2)+A$(3)+A$(4)
110 END
990  '
1000 ' A$(1)
1018 '
1020 DATA 00000000
1030 DATA 00000000
1040 DATA 00000000
1050 DATA 00000000
1060 DATA 00000011
1070 DATA 00000011
1080 DATA 00001111
1090 DATA 00010010
1100 '
1105 ' A$(2)
1108 '
1110 DATA 00110010
1120 DATA 01111111
1130 DATA 11111111
1140 DATA 00011100
1150 DATA 00001000
1160 DATA 00000000
1170 DATA 00000000
1180 DATA 00000000
1190 '
1195 ' A$(3)
1198 '
1200 DATA 00000000
1210 DATA 00000000
1220 DATA 00000000
1230 DATA 00000000
1240 DATA 11000000
1250 DATA 11000000
1260 DATA 11110000
1270 DATA 01001000
1280 '
1285 ' A$(4)
1288 '
1290 DATA 01001100
1300 DATA 11111110
1310 DATA 11111111
1320 DATA 01110000
1330 DATA 00100000
1340 DATA 00000000
1350 DATA 00000000
1360 DATA 00000000
```



A l'affichage, le caractère résultant (un vaisseau spatial) sera formé des quatre caractères dans l'ordre suivant:




REMARQUES:

— Suivant que l'on choisisse un format 8 × 8 ou 16 × 16, la longueur maximum de la chaîne donnée en argument de SPRITE$ varie de 8 à 32 caractères. Tous n'ont pas besoin d'être entrés: MSX-BASIC complète automatiquement la fin de la chaîne par des CHR$(0).

— Chaque sprite est défini par son numéro N pour toutes les instructions d'affichage du MSX-BASIC.

— L'option "double taille" de SCREEN ne modifie que l'aspect à l'écran des sprites.

— Il est possible d'afficher des sprites sur les deux écrans graphiques haute et basse-résolution, mais le contenu de SPRITE$ est toujours défini en fonction de la haute-résolution (256 × 192 points élémentaires sur un écran).

## PUT SPRITE

*Syntaxe*: PUT SPRITE /N° PLAN/ [,[STEP](X,Y)]
[,/Couleur/] [,/N° SPRITE/]

*Application*: Affichage d'un sprite sur l'écran graphique MSX, au point de coordonnées X,Y. Le caractère doit avoir été préalablement défini à l'aide de la variable réservée SPRITE$(N).

Le MSX-BASIC utilise 32 écrans fictifs appelés "PLANS" pour faire apparaître les sprites. Sur chaque plan, un seul sprite peut être affiché à la fois, dans une couleur et à un emplacement quelconque. En revanche, tous les plans sont affichés simultanément: Il est donc possible de faire apparaître en même temps 32 sprites sur l'écran graphique MSX! Quelques règles simples doivent être connues pour utiliser PUT SPRITE:

— /N° PLAN/ représente l'écran fictif employé pour l'affichage, et doit être compris entre 0 et 31.

— Si STEP est omis, le coin supérieur gauche du sprite se place au point de coordonnées absolues X,Y avec X pour abscisse et Y pour ordonnée. Dans le cas contraire, X et Y servent à définir un déplacement sur l'échelle des coordonnées, relativement à la position courante du curseur graphique (voir PSET). Si aucun couple de coordonnée n'est spécifié dans la syntaxe, la position courante du curseur est prise par défaut pour l'affichage.

| | |
|---|---|
| `PUT SPRITE 1,(100,50),,2` | Affiche SPRITE$(2) sur le plan numéro 1, au point de coordonnées (100,50). |
| `PUT SPRITE 1,STEP(10,0),,2` | Affiche SPRITE$(2) sur le plan 1, dix points à droite de la position courante du curseur graphique. |
| `PUT SPRITE 2,,,2` | Affiche SPRITE$(2) sur le plan 2, à la position du curseur. |

— /Couleur/ sert à définir la couleur du sprite. Si elle est omise, la couleur de l'avant plan sélectionnée dans COLOR est prise par défaut.

— /N° SPRITE/ représente le numéro choisi pour le sprite avec SPRITE$(N). Suivant la taille définie par SCREEN pour les sprites, ce numéro peut varier entre 0 et 255 (8 × 8) ou entre 0 et 63 (16 × 16). S'il est omis, /N° PLAN/ est pris par défaut.

| | |
|---|---|
| `PUT SPRITE 2,(10,50),5,4` | Affiche SPRITE$(4) sur le plan 2, au point de coordonnées (10,50) et dans la couleur 5. |
| `PUT SPRITE 2` | Affiche SPRITE$(2) sur le plan 2, à la position du curseur et dans la couleur en vigueur. |

REMARQUES :

— L'affichage des sprites est totalement indépendant du contenu de l'écran.

— Les plans se superposent dans l'ordre de la numérotation : Le plan 1 est devant le plan 5, etc...

— L'abscisse X peut être comprise entre −32 et 255.

— L'ordonnée Y peut être comprise enre −32 et 191. Deux cas particuliers correspondent à des fonctions de PUT SPRITE :

- Si Y=208, tous les plans dont le numéro est supérieur à la valeur spécifiée dans l'instruction disparaissent, jusqu'à ce qu'une nouvelle valeur soit donnée à Y.

- Si Y=209, le plan spécifié (et par conséquent le sprite qui lui est associé) disparaît. Cette possibilité facilite le clignotement et l'animation.

— Si plusieurs sprites doivent coïncider sur l'axe des Y, ou encore se mouvoir à proximité les uns des autres, il faut limiter à 4 le nombre de sprites affichés simultanément.

***Exemple*** : On peut ajouter au programme d'exemple du chapitre 5.2.3 un mobile animé simple créé à l'aide de SPRITE***$*** :

```
10 ' INITIALISATIONS
20 '
30 KEY OFF
40 COLOR 15,5,1
50 SCREEN 2,0
60 '
70 ' DESSIN DU CIEL
80 '
90 FOR N%=1 TO 200
100 X=RND(1)*255:Y=RND(1)*191
110 PSET(X,Y),11
120 NEXT
130 '
140 ' DESSIN DES PLANETES
150 '
160 R=3.141593=/180
170 CIRCLE (255,10),50,10
180 PAINT (255,10),10,10
190 CIRCLE (255,10),70,14,R*130,,.25
200 CIRCLE (255,10),80,14,R*100,,.35
210 PAINT (255,35),14,14
220 CIRCLE (40,30),20,9
230 PAINT (40,30),9,9
240 CIRCLE (120,100),20,11,R*100,R*265,1
250 CIRCLE (140,100),30,11,R*140,R*225,1
260 PAINT (105,100),11,11
270 CIRCLE (20,160),20,8,,,1
280 PAINT (18,150),8,8
290 CIRCLE (40,160),8,8,,,1
300 CIRCLE (155,370),200,3,R*50,R*130,1
310 PAINT (155,185),3,3
320 '
330 ' AFFICHAGE DU TEXTE
340 '
350 OPEN "GRP:" FOR OUTPUT AS #1
360 DRAW "BM10,0"
370 PRINT #1,"L'UNIVERS DU MSX"
380 CLOSE
390 '
400 ' LECTURE DES DATAS DANS A$
410 '
```

```
430 X$="":A$=""
450 FOR N%=1 TO 8
460 READ X$
470 A$=A$+CHR$(VAL("&B"+X$))
480 NEXT N%
500 '
510 ' CONSTITUTION DES SPRITES
520 '
530 SPRITE$(5)=A$
540 '
550 ' BOUCLE GERANT LES DEPLACEMENTS
560 '
570 X1=15:X2=245:K=2
580 FOR X=X1 TO X2 STEP K
590 Y=SIN(T)*30+50+X*.25
610 PUT SPRITE 5,(X,Y),10,5
750 T=T+R*18:NEXT
760 '
770 ' CHANGEMENT DE DIRECTION
780 '
790 SWAP X1,X2:K=-K
840 '
850 ' BOUCLE INFINIE
860 '
870 GOTO 580
880 '
890 ' DESSIN DE L'INVADER
900 '
910 DATA 00100100
920 DATA 00011000
930 DATA 00111100
940 DATA 01100110
950 DATA 11011011
960 DATA 01111110
970 DATA 00100100
980 DATA 00000000
```

| | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | ■ | | | ■ | | |
| 2 | | | | ■ | ■ | | | |
| 3 | | | ■ | ■ | ■ | ■ | | |
| 4 | | ■ | ■ | | | ■ | ■ | |
| 5 | ■ | ■ | | ■ | ■ | | ■ | ■ |
| 6 | | ■ | ■ | ■ | ■ | ■ | ■ | |
| 7 | | | ■ | | | ■ | | |
| 8 | | | | | | | | |

**Explications :**

— 10-400 : Ces lignes sont identiques au programme du paragraphe 5.2.4. L'instruction SCREEN de la ligne 10 sélectionne le mode graphique haute-résolution, et une taille 8 × 8 pour les sprites.

— 400-480 : Chargement dans la chaîne A$ des huit lignes de DATA constituant le sprite invader (lignes 890,980).

— 500-540 : SPRITE$(5) est initialisé avec A$.

— 550-880 : Constitution d'une boucle infinie gérant le déplacement du sprite. Le contenu de cette boucle peut être détaillé :

— 570: Valeur limite pour les coordonnées d'affichage du sprite.

— 580: Début de la boucle faisant varier X dans ces valeurs limites.

— 590: L'ordonnée Y varie avec X, en décrivant un tracé sinusoïdal fonction de T. Au premier passage, T=0.

— 610: Affichage de SPRITE$(5) sur le plan numéro 5, au point X,Y et dans la couleur 10.

— 750: T est incrémenté en fonction de R (voir ligne 160), puis retour dans la boucle.

— 770-870: Une fois la première boucle effectuée le mobile repart au début, mais en sens inverse.

## ON SPRITE GOSUB

*Syntaxe*: ON SPRITE GOSUB /N° de ligne/

*Application*: Il est possible de saisir la "collision" de deux sprites sur l'écran pour effectuer un branchement prioritaire vers un sous-programme avec ON SPRITE GOSUB. Comme pour les autres types de déroutements, l'instruction SPRITE ON valide cette déclaration.

REMARQUES:

— Chaque fois qu'un déroutement est validé, un SPRITE STOP est automatiquement exécuté. En retour de sous-programme, un SPRITE ON supprime l'inhibition des déroutements.

— ON SPRITE GOSUB s'utilise principalement dans les jeux: Il permet la saisie d'événements en temps réel du type collision de vaisseaux spatiaux, ennemi battu, etc...

— Les déroutements ne fonctionnent qu'en mode programme.

*Voir aussi*: SPRITE ON/OFF/STOP.

## SPRITE ON/OFF/STOP

*Syntaxe*: SPRITE ON
SPRITE OFF
SPRITE STOP

*Applications*:

— SPRITE ON valide le déroutement par la déclaration ON SPRITE GOSUB.

— SPRITE OFF supprime la prise en compte de ON SPRITE GOSUB.

— SPRITE STOP mémorise la rencontre de deux sprites, mais ne valide le déroutement qu'après rencontre de SPRITE ON (voir exemple).

### 5.3.3. Le programme terminé

Nous en savons maintenant assez sur la gestion des sprites pour compléter et terminer le programme d'exemple qui nous a servi de guide tout au long de ce chapitre:

```
10 ' INITIALISATIONS
20 '
30 KEY OFF
40 COLOR 15,5,1
50 SCREEN 2,3
60 '
70 ' DESSIN DU CIEL
80 '
90 FOR N%=1 TO 200
100 X=RND(1)*255:Y=RND(1)*191
110 PSET(X,Y),11
120 NEXT
130 '
140 ' DESSIN DES PLANETES
150 '
160 R=3.141593=/180
170 CIRCLE (255,10),50,10
180 PAINT (255,10),10,10
190 CIRCLE (255,10),70,14,R*130,,.25
200 CIRCLE (255,10),80,14,R*100,,.35
210 PAINT (255,35),14,14
220 CIRCLE (40,30),20,9
230 PAINT (40,30),9,9
240 CIRCLE (120,100),20,11,R*100,R*265,1
250 CIRCLE (140,100),30,11,R*140,R*225,1
260 PAINT (105,100),11,11
270 CIRCLE (20,160),20,8,,,1
280 PAINT (18,150),8,8
290 CIRCLE (40,160),8,8,,,1
300 CIRCLE (155,370),200,3,R*50,R*130,1
310 PAINT (155,185),3,3
320 '
330 ' AFFICHAGE DU TEXTE
340 '
350 OPEN "GRP:" FOR OUTPUT AS #1
360 DRAW "BM10,0"
370 PRINT #1,"L'UNIVERS DU MSX"
380 CLOSE
390 '
400 ' LECTURE DES DATAS DANS A($)
410 '
420 DIM A$(5)
440 FOR M%=1 TO 5
450 FOR N%=1 TO 8
460 READ X$
470 A$(M%)=A$(M%)+CHR$(VAL("&B"+X$))
```

```
480 NEXT N%
490 NEXT M%
500 '
510 ' CONSTITUTION DES SPRITES
520 '
530 SPRITE$(5)=A$(1):SPRITE$(6)=A$(2)+A$(3)+A$(4)+A$(5)
540 '
550 ' BOUCLE GERANT LES DEPLACEMENTS
560 '
570 X1=15:X2=245:K=2
580 FOR X=X1 TO X2 STEP K
590 Y=SIN(T)*30+50+X*.25
600 Y1=80+X*.8-30
610 PUT SPRITE 5,(X,Y),10,5
620 '
630 ' CLIGNOTEMENT DU VAISSEAU SPATIAL
640 '
650 IF Y1>191 THEN 710
660 PUT SPRITE 6,(X*1.5,Y1),6,6
670 FOR L=1 TO 20:NEXT L
680 IF Y1<35 OR Y1>150 THEN 710
690 PUT SPRITE 6,(X*1.5,Y1),15,6
700 FOR L=1 TO 20:NEXT L
710 '
720 ' SONORISATION DU MOUVEMENT
730 '
740 IF (X MOD 5)=1 THEN PLAY "V15O5L64AF"
750 T=T+R*18:NEXT
760 '
770 ' CHANGEMENT DE DIRECTION
780 '
790 SWAP X1,X2:K=-K
800 '
810 ' SONORISE LE CHANGEMENT
820 '
830 PLAY "V15O5L64ABCDEF"
840 '
850 ' BOUCLE INFINIE
860 '
870 GOTO 580
880 '
890 ' DESSIN DE L'INVADER SPRITE$(5)
900 '
910 DATA 00100100
920 DATA 00011000
930 DATA 00111100
940 DATA 01100110
950 DATA 11011011
960 DATA 01111110
970 DATA 00100100
980 DATA 00000000
990  '
1000 ' DESSIN DU VAISSEAU SPRITE$(6)
```

```
1010 '
1020 DATA 00000000
1030 DATA 00000000
1040 DATA 00000000
1050 DATA 00000000
1060 DATA 00000011
1070 DATA 00000011
1080 DATA 00001111
1090 DATA 00010010
1100 '
1110 DATA 00110010
1120 DATA 01111111
1130 DATA 11111111
1140 DATA 00011100
1150 DATA 00001000
1160 DATA 00000000
1170 DATA 00000000
1180 DATA 00000000
1190 '
1200 DATA 00000000
1210 DATA 00000000
1220 DATA 00000000
1230 DATA 00000000
1240 DATA 11000000
1250 DATA 11000000
1260 DATA 11110000
1270 DATA 01001000
1280 '
1290 DATA 01001100
1300 DATA 11111110
1310 DATA 11111111
1320 DATA 01110000
1330 DATA 00100000
1340 DATA 00000000
1350 DATA 00000000
1360 DATA 00000000
```

**Explications :**

Ce programme, peu différent des précédents gère le déplacement de deux mobiles, un invader et un vaisseau spatial (SPRITE$(5) et SPRITE$(6)). La taille choisie pour les sprites est le maximum disponible, c'est-à-dire 16 × 16 avec l'option double taille (ligne 50).

Le clignotement du vaisseau est réalisé en affichant alternativement SPRITE$(6) dans deux couleurs différentes (lignes 630,700).

Le mouvement est sonorisé par l'instruction PLAY, aux lignes 740 et 830 (voir chapitre 6, le son MSX).

# 6. Le son du MSX

## 6.1. GÉNÉRALITÉS

Munis d'une "puce" spécialisée chargée de la gestion du son, les ordinateurs MSX présentent de ce fait des dispositions très au-dessus de la moyenne en ce domaine : On peut programmer sur ce mini-synthétiseur l'enveloppe, la hauteur et la longueur des sons et des notes sur trois canaux indépendants, puis les mélanger à un bruit blanc. Pour jouir à loisir de vos créations sonores, une prise aux normes RCA permet le raccordement à une chaîne haute-fidélité.

Pour créer et mémoriser une mélodie ou un son, l'instruction PLAY utilise un macro-langage très puissant similaire à celui de l'instruction graphique DRAW. On peut par son intermédiaire générer des accords,

changer le volume, et modifier tous les paramètres d'un son, y compris son enveloppe. Comme pour DRAW, la possibilité d'exécuter des sous-chaînes permet la création de mélodies d'une longueur quasi-illimitée !

Il est également possible d'accéder directement aux registres du synthétiseur par l'instruction Basic SOUND. Plus complexes à mettre en œuvre, elle autorise la création de tous les types de son par un contrôle plus précis de l'enveloppe, et la modification des paramètres par défaut de l'instruction PLAY.

## 6.2. LES COMMANDES DE GESTION DU SON

### BEEP

*Syntaxe*: BEEP

*Application*: Génère un son de longueur fixe.

REMARQUES:

— BEEP est équivalent à PRINT CHR$(7).

— BEEP réinitialise PLAY avec les valeurs par défaut.

### SOUND

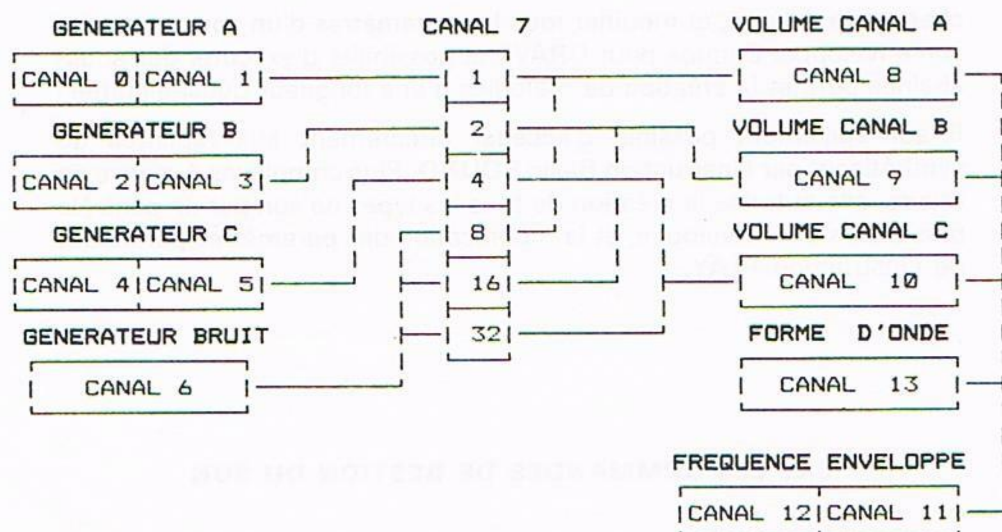
*Syntaxe*: SOUND /N° CANAL/,N

*Application*: SOUND permet l'accès direct aux 14 registres du circuit son MSX:

— /N° CANAL/ correspond au numéro de registre, et doit être compris entre 0 et 13.

— N est la valeur à entrer dans le registre, dont les valeurs limites dépendent du canal choisi. En aucun cas N ne peut être supérieur à 255.

Avant d'aborder le fonctionnement de l'instruction SOUND, il est nécessaire de détailler l'architecture interne du circuit intégré permettant de générer les sons:




Pour créer un son par SOUND, il faut spécifier quels générateurs vont être utilisés, puis choisir le volume et l'enveloppe du son. Le son est émis après définition de la fréquence de chacun d'eux:

— La fréquence des trois générateurs de son A, B et C est codée sur deux canaux, (0 et 1) pour le A, (2 et 3) pour le B, (4 et 5) pour le C. Si N0, N1, N2, N3, N4 et N5 représentent les valeurs de N associées aux 6 canaux, le calcul de la fréquence des trois générateurs s'effectue de la manière suivante:

```
GENERATEUR A          F = (N0*255)+N1

GENERATEUR B          F = (N2*255)+N3

GENERATEUR C          F = (N4*255)+N5
```

N0? N2 et N4 doivent être compris entre 0 et 15.

N1, N3 et N5 doivent être compris entre 0 et 255.

Ce principe de codage sur deux octets autorise des valeurs allant jusqu'à 4080 Hz pour la fréquence des sons.

***Exemples***:

| | |
|---|---|
| `SOUND 0,2:SOUND 1,200` | Règle la fréquence du générateur A sur 700 Hz. |
| `SOUND 4,0:SOUND 5,200` | Règle le générateur C sur 200 Hz. |

— Le canal 6 sert uniquement à régler le niveau du générateur de bruit: N doit être compris entre 0 et 63.

***Exemples*** :

| | |
|---|---|
| `SOUND 6,40` | Règle le niveau du générateur de bruit sur 40. |
| `SOUND 6,0` | Supprime l'émission de bruit. |

— Le canal 7 est la "voie de passage" pour le générateur de bruit et les trois oscillateurs : Pour se retrouver en sortie, un son doit être validé par ce registre. Les générateurs actifs sont sélectionnés suivant la valeur prise par chacun des bits qui composent l'octet envoyé sur le canal 7 :

Les trois bits de poids 1, 2 et 4 conditionnent la sortie pour les trois oscillateurs A, B et C. Le son de l'un d'entre eux n'est transmis que si le bit correspondant est à 0.

Les trois bits de poids 8, 16 et 32 autorisent le mélange du son émis par le générateur de bruit avec les trois oscillateurs A, B et C. Comme précédemment, le mélange n'est effectif que si le bit correspondant à la voie sélectionnée est à 0.

***Exemples*** :

Pour utiliser ce canal, il est préférable de rentrer directement le nombre N en binaire :

| | |
|---|---|
| `SOUND 7,&B111000` | Valide la sortie pour les trois oscillateurs, mais interdit le passage du bruit. |
| `SOUND 7,&B110110` | Seul le son du générateur mélangé à du bruit est transmis à la sortie. |
| `SOUND 7,&B000000` | Les trois canaux sont mélangés au bruit et transmis vers la sortie. |

— Les trois canaux 8, 9 et 10 règlent le volume des trois voies de sortie du générateur de sons : Le canal 8 est associé à l'oscillateur A, le canal 9 à l'oscillateur B et le canal 10 à l'oscillateur C. Si le son d'un oscillateur est mélangé à du bruit, c'est le volume du mélange qui est modifié. La valeur du volume N doit être comprise entre Ø et 16.

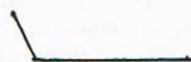
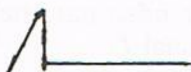
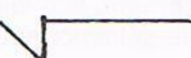
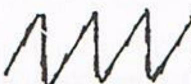
***Exemples*** :

| | |
|---|---|
| `SOUND 8,15` | Règle le volume de sortie au maximum pour le générateur A. |

Si N=16, le volume de sortie est contrôlé par le générateur d'enveloppe (voir canal 13) :

| | |
|---|---|
| `SOUND 8,16:SOUND 9,15` | Le volume de sortie de l'oscillateur A est contrôlé par le générateur d'enveloppe, celui du B est réglé à sa valeur maximum. |

— Le canal 13 permet la sélection d'une forme d'onde servant d'enveloppe pour les trois sorties : Le volume de sortie est modulé par l'enveloppe choisie. Huit formes d'ondes sont disponibles, en fonction de la valeur bit à bit du nombre N (N doit être compris entre 0 et 15).

| BITS : | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|
| N= | 0 | 0 | X | X |  |
| | 1 | 0 | 0 | 1 | |
| N= | 0 | 1 | X | X |  |
| | 1 | 1 | 1 | 1 | |
| N= | 1 | 0 | 0 | 0 |  |
| N= | 1 | 0 | 1 | 0 |  |
| N= | 1 | 0 | 1 | 1 |  |
| N= | 1 | 1 | 0 | 0 |  |
| N= | 1 | 1 | 0 | 1 |  |
| N= | 1 | 1 | 1 | 0 |  |

***Exemple*** :

`SOUND 13,&B1110` Sélectionne une forme d'onde triangulaire pour l'enveloppe du son.

— Les canaux 11 et 12 permettent le réglage de la période de cette enveloppe. Cette période doit être comprise entre 0 et 65535.

Soit N11 et N12 les valeurs saisies par SOUND pour les canaux 11 et 12, la période P est égale à :

P=(N12*256)+N11 N11 et N12 sont comprises entre 0 et 255.

***Exemple*** :

`SOUND 11,255:SOUND 12,0` Règle la période sur 255.

Frequence = 124000 / période

REMARQUES :

— N peut être une constante ou une expression numérique.

— Une fois la valeur de N spécifiée pour un registre, elle reste valide jusqu'à exécution d'une nouvelle instruction SOUND.

— La modification d'un paramètre par SOUND est également valable pour les valeurs par défaut de l'instruction PLAY.

— SOUND permet très facilement la génération de bruits aléatoires (voir exemple).

***Exemple*** :

```
10 ' BRUITS ALEATOIRES
15 FOR I=0 TO 13
30 SOUND I,0
40 NEXT
50 SOUND 7,254:SOUND 8,15
60 FOR I=1 TO 255 STEP 0.1
70 R=INT(RND(1)*I)
80 SOUND 0,R
90 NEXT
```

## PLAY

*Syntaxe* : PLAY "/Chaîne 1/" [,"/Chaîne 2/"] [,"/Chaîne 3/"]

*Application* : Émission d'une mélodie inclue dans les chaînes données en argument sous forme de commandes à une lettre. Les trois chaînes sont indépendantes, elles correspondent aux trois canaux du synthétiseur MSX (voir SOUND).

Chacune d'elle peut contenir jusqu'à 255 caractères maximum. 16 commandes différentes sont reconnues par l'instruction PLAY. Elles sont interprétées de gauche à droite à l'intérieur de chaque chaîne. Des valeurs par défaut permettent l'émission de notes sans avoir à redéfinir tous les paramètres d'un son :

**A,B,C,D,E,F,G** Émission d'une note en fonction de la lettre entrée : A pour LA, B pour SI, C pour DO, D pour RE, E pour MI, F pour FA et G pour SOL.
Si la lettre est suivie du signe "#" ou d'un "+", le dièse de la note est émis. Si la lettre est suivie du signe "−", c'est le bémol qui est émis. Ces deux options ne sont disponibles que pour les notes correspondant aux touches noires du clavier d'un piano : B+ provoque une erreur.

**O** N — Règle l'octave en cours pour toutes les notes émises après la commande. 8 octaves numérotés de 1 à 8 sont disponibles sous MSX-BASIC, chaque octave allant de DO à SI. La valeur par défaut pour N est de 4.

**N** N — Émet une note N. N peut être inclu entre 0 et 96, ce qui correspond aux huit octaves du MSX. Cette commande permet un contrôle total de la hauteur du son, demi-ton par demi-ton, sans référence à une note de la gamme. Si N=0, un silence de la longueur en cours est généré (voir L). Le DO de l'octave 4 est équivalent à N=36.

**L** N — Règle la longueur des notes émises par PLAY. N peut être compris entre 1 pour la note la plus longue et 64 pour la plus courte. Le rapport entre la longueur maximale et la longueur choisie pour la note est de 1/N : L4 est quatre fois plus courte que L1. La valeur par défaut pour L est de 4 ce qui correspond à une note "noire". Si l'on veut changer uniquement la durée d'une note sans modifier la longueur en cours, il suffit de faire suivre le nom de la note par la durée choisie : L32A et A32 ont un effet similaire.

**R** N — Émet un silence de longueur N. N peut être compris entre 1 et 64, et suit les mêmes règles que pour L. La valeur par défaut est de 4.

**T** N — Fixe le nombre de noires émises à la minute, et par conséquent le tempo de la mélodie. N peut être compris entre 32 et 255. La valeur par défaut est 120. Un point inséré à la site d'une note multiplie sa longueur par 3/2 (note pointée). Si plusieurs points sont entrés successivement, la longueur de la note est modifiée en conséquence : "A..." correspond à une multiplication par 27/8 de la longueur de "A". Des points peuvent être insérés à la suite d'une pause **R** pour modifier sa durée.

**V** N — Règle le volume de sortie, compris entre 0 et 15. Si N=0, aucun son n'est émis. N=8 est pris par défaut.

**M** N — Sélectionne la fréquence de l'enveloppe, avec N compris entre 1 et 65535. Les règles d'utilisation sont identiques à celles qui ont été définies pour l'instruction SOUND. Par défaut, N=255.

**S** N — Choix d'une forme d'onde pour l'enveloppe du son: Comme pour SOUND, N peut valoir de 0 à 15 et permet le choix de huit formes différentes (voir SOUND). Par défaut, N=1.

**X** A$ — La commande X permet d'exécuter une sous-chaîne à l'intérieur d'une des constantes chaînes données en argument de PLAY. Le point-virgule en fin d'expression est obligatoire. L'insertion de plusieurs sous-chaînes à l'intérieur d'une même chaîne permet de créer des mélodies de longueur supérieure à 255 caractères.

***Exemples***:

| | |
|---|---|
| `PLAY "CDEFGAB"` | Joue la gamme sur sept notes. |
| `PLAY "C+D+F+G+A+"` | Joue les cinq dièses de la gamme. |
| `PLAY "O2CDEFO4GAB"` | Joue la gamme avec montée de deux octaves à partir du SOL. |
| `PLAY "O1AO3AO5AO7A"` | Émet un LA à quatre hauteurs différentes. |
| `PLAY "N35N36N37N38"` | Émission de quatre notes séparées par un demi-ton. |
| `PLAY "L8AL6AL4AL2A"` | Multiplication par 2 de la longueur de la note à chaque émission. L modifie la longueur de toutes les notes qui suivent la commande. Pour ne modifier que A, il faut écrire: PLAY "A8A6A4A2". |
| `PLAY "T300ABCDE"` | Sélection d'un tempo rapide. |
| `PLAY "V5CV10CV15C"` | Montée progressive du volume. |
| `PLAY "S2M300CDE"` | Modification de l'enveloppe du son. |
| `10 A$="ABC":B$="EFG"`<br>`20 PLAY "XA$;XB$;"` | Exécution de deux sous-chaînes A$ et B$. |

REMARQUES:

— Dans toutes les commandes, l'argument N peut être une constante numérique ou l'expression **=/Variable/;** dans laquelle /Variable/ représente le nom d'une variable numérique. Comme pour **X**, le point-virgule (;) est obligatoire en fin d'expression lorsqu'on utilise une variable de cette façon:

```
10 ' Emission des 96 tonalités du MSX-BASIC
20 '
10 FOR P=1 TO 96
20 PLAY "N=P;"
30 NEXT P
```

```
10 ' Variation simultanée de tous les paramètres
20 '
25 FOR O%=3 TO 6:'                               OCTAVE
30 FOR T%=120 TO 240 STEP 120:'                  TEMPO
40 FOR L%=16 TO 64 STEP 48:'                     LONGUEUR
50 FOR V%=10 TO 15 STEP 5:'                      VOLUME
55 '
60 PLAY "T=T%;V=V%;O=O%;L=L%;ABCDEFGFEDCBA"
65 '
70 NEXT V%
80 NEXT L%
90 NEXT T%
95 NEXT O%
```

En utilisant la commande **X**, la ligne 60 peut se décomposer en:

```
20 A$="ABCDEFGFEDCBA"
60 PLAY "T=T%;V=V%;O=O%;L=L%;XA$;"
```

— La possibilité de donner trois chaînes séparées par des virgules en argument de PLAY permet de créer des accords sur trois notes:

`PLAY "C","E","G"` Émet l'accord DO-MI-SOL.

`PLAY "O2CDE","O6CDE"` Joue 2 octaves en même temps.

On peut se servir des trois canaux du générateur MSX pour faire jouer simultanément des mélodies différentes par PLAY. Dans le programme qui suit, on utilise également les commandes M et S pour modifier l'enveloppe du son:

```
10 ' Emission simultanée de deux mélodies
20 '
25 A$="M50S14":B$="ABCDEFGFEDCBA":C$="ABCDEF"
30 '
35 FOR O%=3 TO 6:'                               OCTAVE
40 FOR T%=120 TO 240 STEP 120:'                  TEMPO
50 FOR L%=16 TO 64 STEP 48:'                     LONGUEUR
60 FOR V%=10 TO 15 STEP 5:'                      VOLUME
65 '
70 PLAY "XA$;T=T%;V=V%;O=O%;L=L%;XB$;","XC$;"
75 '
80 NEXT V%
90 NEXT L%
95 NEXT T%
98 NEXT O%
```

*Voir aussi*: SOUND (/N° Voie/)

**PLAY(N)**

*Syntaxe*: PLAY (/N° VOIE/)

*Application*: Cette fonction permet de savoir si l'une des trois mélodies incluses sous forme de chaînes dans une instruction PLAY est en cours d'exécution ou terminée. /N° VOIE/ est associé aux trois canaux du générateur de son MSX. Il peut être compris entre 0 et 3:

— Si /N° VOIE/ est égal à 1, 2 ou 3, la fonction renvoie −1 si le canal est concerné est encore en opération et 0 dans le cas contraire.

— Si /N° VOIE/ est égal à 0, le résultat est − 1 si n'importe lequel des canaux est en opération, et 0 dans le cas contraire.

REMARQUE: Immédiatement après la rencontre d'une instruction PLAY dans un programme, la fonction renvoie − 1 quelque soit le contenu des trois chaînes.

*Voir aussi*: PLAY.

Frequence = 124000/periode

# Appendice 1

# Le jeu de caractères MSX



# Appendice 2

# Les messages d'erreur du MSX-BASIC

Toute erreur pouvant survenir dans un programme (ou en mode direct), qu'elle soit due à sa rédaction (syntaxe ou logique erronée, instructions manquantes), ou produite par l'exécution elle-même (dépassement de capacité, etc...) est traitée par le MSX-BASIC. Celui-ci interrompt alors l'exécution, et affiche un message d'erreur approprié selon la syntaxe suivante :

XXXX in [Nx de ligne]

Dans laquelle XXXX est un message définissant le type de l'erreur, dont la liste est donnée ci-dessous, et [Nx de ligne] est la ligne ou l'erreur s'est produite.

Parallèlement, et si la commande "ON ERROR GOTO" figure au début du programme, les variables réservées ERR et ERL contiendront respectivement le numéro de l'erreur, et le numéro de la ligne où l'erreur s'est produite. Voyez à ce sujet le paragraphe 3.3.6.

| *Code* | *Message* |
| --- | --- |
| 1 | **NEXT without FOR**<br>Une commande NEXT a été rencontrée, sans instruction FOR correspondante. |
| 2 | **Syntax error**<br>Une ligne de programme ou une commande directe est formulée de manière erronée. |
| 3 | **RETURN without GOSUB**<br>Un RETURN est rencontré, sans que le branchement ait été préalablement effectué par GOSUB. |
| 4 | **Out of DATA**<br>La commande READ est exécutée, alors que les lignes de DATA ont été lues en entier. |
| 5 | **Illegal function call**<br>Un paramètre hors limites a été passé à une instruction ou une fonction. Cette erreur peut aussi être provoquée par:<br>— Un indice négatif ou trop élevé,<br>— Un argument négatif ou nul à LOG,<br>— Un argument négatif à SQR. |
| 6 | **Overflow**<br>Les calculs ont provoqué un dépassement de capacité arithmétique. |
| 7 | **Out of memory**<br>Il n'y a plus de mémoire disponible:<br>— programme trop important,<br>— trop de boucles FOR/NEXT ou GOSUBS imbriqués,<br>— trop de fichiers ouverts en même temps,<br>— trop de variables ou expressions complexes à évaluer. |
| 8 | **Undefined line number**<br>Un numéro de ligne inexistant figure dans un GOTO, GOSUB ou IF-THEN-ELSE. |

**9** **Subscript out of range**

Un élément de tableau est appelé avec un indice trop élevé (supérieur à ce qui a été déclaré dans DIM), ou avec un nombre d'indices erroné.

**10** **Redimensionned array**

Une tentative est faite de redimensionner un tableau figurant déjà dans une instruction DIM.

**11** **Division by zero**

Une division par zéro est rencontrée dans une expression.

**12** **Illegal direct**

On a tenté d'utiliser en mode direct une instruction ne pouvant figurer que dans un programme (par ex. INPUT, READ, etc...).

**13** **Type mismatch**

On a tenté d'attribuer une valeur numérique à une variable-chaîne, ou vice-versa. Peut aussi être provoqué par l'attribution d'un argument de type erroné à une fonction: par ex. SIN(A$).

**14** **Out of string space**

Le nombre des variables-chaîne a provoqué l'épuisement de la mémoire.

**15** **String too long**

Tentative de créer une chaîne de longueur supérieure à 255 caractères.

**16** **String formula too complex**

Une expression traitant des chaînes de caractères (MID$, LEFT$, LEN, etc...) est trop longue ou trop complexe.

**17** **Can't continue**

Une tentative est faite de reprendre un programme par CONT, alors que:

— l'arrêt était dû à une erreur,
— le programme a été modifié pendant la pause,
— le programme n'existe pas.

**18** **Undefined user function**

Une fonction utilisateur FN est appelée avant d'avoir été définie par DEF FN.

**19** **Device I/O error**

Une erreur d'entrée-sortie s'est produite, pendant le fonctionnement du magnétocassette, de l'imprimante ou de la vidéo. Le Basic est incapable de pallier à l'erreur.

**20** **Verify error**

Après sauvegarde, le programme sauvegardé sur cassette est différent du programme en mémoire.

**21** **No RESUME**

Un branchement s'est effectué vers une routine de traitement d'erreurs, mais celle-ci ne se termine pas par RESUME.

**22** **RESUME without error**

Un RESUME est rencontré, sans que le branchement ait été provoqué par la détection d'une erreur.

**23** **Unprintable error**

Aucun message n'existe pour l'erreur qui vient de se produire. Peut être provoqué par une commande ERROR suivie d'un code non défini.

**24** **Missing operand**

Un opérateur (arithmétique, logique ou relationnel) sans opérande correspondante est rencontré dans une expression.

**25** **Line buffer overflow**

Une ligne tapée (ou éditée) contient trop de caractères.

**26-49** **ERREURS NON DEFINIES**

Ces codes devraient être laissés libres par l'utilisateur, afin de rester disponibles pour des extensions futures.

REMARQUE: Plusieurs des erreurs qui suivent, concernant les fichiers à accès direct, sont spécifiques à l'extension DISQUE du MSX-BASIC. Celles-ci sont repérées par un astérisque (*).

**50** **FIELD overflow** *

Le nombre d'octets déclaré dans un FIELD est supérieur à la longueur d'enregistrement par défaut, ou à celle spécifiée dans l'OPEN correspondant.

**51** **Internal error**

Une erreur de fonctionnement interne s'est produite.

**52** **Bad file number**

Une instruction se réfère à un fichier non ouvert, ou hors des limites spécifiées par MAXFILES.

**53** **File not found** *

Une instruction LOAD, KILL ou OPEN se réfère à un fichier inexistant.

**54** **File already open**

Tentative d'utiliser les instructions OPEN ou KILL sur un fichier déjà ouvert par OPEN.

**55** **Un INPUT#** est exécuté sur un fichier vide, ou sur un fichier séquentiel alors que tout son contenu a déjà été lu en totalité.

**56** **Bad file name**

Un nom de fichier incorrect est spécifié dans un LOAD, SAVE, OPEN, etc...

**57** **Direct statement in file**

Une structure incorrecte a été rencontrée lors du chargement d'un programme par LOAD (commande en mode direct, etc.). Le chargement est interrompu.

**58** **Sequential I/O only** *

Une tentative d'accès direct a été faite sur un fichier séquentiel.

**59** **File not OPEN**

Une tentative d'accès est faite sur un fichier qui n'a pas encore été ouvert par OPEN.

**60** **Bad allocation table** *

La table d'allocation d'une disquette est détruite ou non-conforme.

**61** **Bad file mode** *

Une tentative a été faite d'utiliser PUT, GET ou LOF sur un fichier séquentiel, LOAD sur un fichier à accès direct, ou d'exécuter OPEN avec une option autre que I, O, A ou R.

**62** **Bad drive name** *

Une syntaxe autre que A :, B :, etc. a été utilisée pour nommer un lecteur de disquettes.

**64** **File still open** *

Le fichier doit être refermé par un CLOSE.

**65** **File already exists** *

Le nouveau nom de fichier spécifié dans NAME existe déjà.

**66** **Disk Full** *

La disquette sur laquelle on tente d'enregistrer est pleine.

**67** **Too many files** *

Une tentative est faite de créer un nouveau fichier (SAVE, OPEN, etc.), alors que le répertoire de la disquette est déjà plein.

**68** **Disk write protected** *

Le disque sur lequel on tente d'enregistrer est protégé en écriture.

**69** **Disk I/O error** *

Une erreur d'entrée/sortie non définie s'est produite pendant un accès disque.

**70-255** **CODES D'ERREURS NON ATTRIBUÉS**

Ces codes sont disponibles pour la définition de vos propres codes d'erreur, à l'aide de l'instruction ERROR.

# Appendice 3

# Caractéristiques techniques du standard MSX

**Microprocesseur**

**Type** : Z–80A
**Fréquence :** 3,58 MHz

**Mémoire Morte (ROM)**

MSX-Basic : 32 Ko

**Mémoire vive (RAM)**

Mini : 8 Koctets
Maxi : 64 Koctets
Mémoire vive vidéo : 16 Koctets.

**Affichage**

Mode texte : 24 lignes de 32 caractères
24 lignes de 40 caractères.

Mode graphique : 256 × 192 points.

Basse résolution : 64 × 48blocs.

**Couleurs**

8 pour le premier plan,
8 pour le fond.

**Son**

Synthétiseur incorporé

Nombre de voies : 3
Octaves : 8

**Interface cassette**

Vitesse de sauvegarde variable entre 1200 et 2400 Bauds.

**Clavier**

73 touches
5 touches fonctions à 10 fonctions.

**Connecteurs pour cartouches**

Un ou deux connecteurs de 50 contacts.

**Bus d'extensions**

50 contacts, optionnel.

**Interface imprimante**

Norme parallèle Centronics, sur connecteur spécial.

**Interface manette de jeu**

Un ou deux connecteurs.

**Interface vidéo**

Couleur : RVB (péritel)
Monochrome : prise moniteur

**Interface son**

Sortie ampli sur connecteur RCA standard.

# Index alphabétique

*Imprimé en France.* — JOUVE, 18, rue Saint-Denis, 75001 PARIS
N° 13622. Dépôt légal : décembre 1984
N° d'éditeur : 4194

Basé sur la référence officielle du MSX telle qu'elle a été définie par MicroSoft, ce livre regroupe toutes les instructions du MSX-BASIC illustrées par de nombreux programmes commentés, ainsi que les commandes du MSX-DOS, le système d'exploitation de disquettes du standard. Le classement des instructions par familles permettra à l'utilisateur chevronné de retrouver rapidement les informations dont il a besoin. Le débutant n'a pas été oublié, la première partie de l'ouvrage incluant un cours de programmation portant sur les instructions les plus simples du langage. L'intérêt d'un ouvrage spécialisé MSX réside dans sa portabilité : Toutes les informations et programmes contenus dans ce livre s'appliquent à toutes les machines correspondant au standard.

EYROLLES



8517

Maquette : Maurice Apelbaum